

muraja

ORAO

uvod u rad i programiranje

Damir Muraja

ORAO

UVOD U RAD I PROGRAMIRANJE

Narodna tehnika SR Hrvatske

Zagreb, 1988

UVOD

Mikroračunalo »orao« namijenjeno je u prvom redu za učenja programiranja. To ne znači da se tim mikroračunalom ne mogu obavljati i neki teži zadaci. Ova knjige napisane je s namjenom da poslužI kao priručnik iz kojeg se može naučiti BASIC kojim »orao « raspolaže.

Ne treba se zanositi idajom da se programiranja može naučiti samo čitanjem knjiga. Da biste naučil programirati morate provesti mnogo sati za računalom. Ova knjiga će vam poslužiti da naučite instrukcija kojima «orao« raspolaže i njihovu sintaksu. Nakon toga prostaju vam mnogi sati rade za mikroračunalom i učanja po matodi pokušaja i pogreska. Greška pri radu neka vas ne plaše jer jedino iz takvih grešaka može se «stvarno» shvatiti rad računala.

Jednom je netko lijepo rekao da je osnovna mana računala to što rade sto im naradimo a ne ono što želimo. Kad budete napisali dvadesetak programa i kad ispravite grške koje će se u njima javiti postat će vam jasna ova izreka.

Mikroračunalo »orao« stekło je veliku popularnost u školama. Proizvođač ovog računala odlučio se da unese poboljšanje u BASICU. Zbog toga »orla« srećemo sa dvije verzije BASICA. Prva (stara) verzija nema mnoge komande koje su kasnije drugoj (novoj) verziji dodane. OVA JE KNJIGA NAMIJENJENA KORISNICIMA KOJI IMAJU NOVU VERZIJU BASICA.

Stara verzija BASICA prepoznaje se po tome što mikroračunalo »orao« po uključenju na gornjem rubu ekrana ispiše «« ORAO »». Ako imate ovakvu verziju računala, možete na njoj raditi većinu naredbi iz ove knjige. Ipak bi bilo najbolje da nabavite novu verziju BASICA jer se to svodi na jednostavnu izmjenu sadržaja dva ugrađena čipa ROM 2764. Ovo ćete najlakše učiniti ako se obratite direktno na proizvođača ovog mikroračunala čija je adresa:

P. E. L.
OOUR ELEKTRONIKA
V. Nazora 2
42000 Varaždin

ŠTO MOŽE »ORAO«

Kao što sam već rekao osnovne namjena tog mikroračunala je učenje programiranja. Mikroračunalo »orao« raspolaže grafikom srednje rezolucije (256 x 256 točaka) programabilnim generatorom zvuka mogućnošću rada s magnetskim medijima i dakako setom slova koji obuhvaća i naša latinična slova. Svakim od ovih svojstava mikroračunala »orao« bavit ćemo se u zasebnom poglavlju a kada iskoristimo sve njegove mogućnosti vidjet ćemo da je pred nama mikroračunalo koje se može i te kako korisno upotrijebiti.

Pogledajmo malo neka karakteristike mikroračunala »orao«. Ako vam ovi podaci ništa ne znače slobodno ih preskočite. Kasnije kada naučite više o programiranju sve to za vas će dobiti jasno značenje pa ćete tek onda pročitati ovaj dio.

»Orao« raspolaže aritmetikom kliznog zareza (floating point). Računa se s točnošću od šest znamenki. Brojevi su prikazani u eksponencijalnom obliku najmanji pozitivni broj je 10E–38 a najveći pozitivni broj 10E38.

Dužina stringova je najviše 255 znakova.

Najveća dužina programske linije je 72 znaka. Može je definirati i manje gornja granica za dužinu programske linije.

Broj programske linije je pozitivni cijeli broj veći od 0 ili manji od 64000.

Računalo posjeduje BASIC INTERPRETER koji podržava rad s kasetofonom i ispis na RS232 port.

Moguće je pozivanje strojnih programa iz BASICA.

Računalo je konstruirano oko mikroprocesora 6502 i posjeduje monitor i minisembler koji omogućuju programiranje u mašinskom jeziku. Ovisno o verziji računala posjeduje 32 ili 16 K RAM-a.

Ekran je organiziran kao bitna mapa i ima 256 x 256 točaka. Na postoji tekst ekran i sav se tekst ispisuje u grafičkom ekranu u matrici 8 x 8 odnosno 32 reda sa po 32 znaka u redu.

PRIKLJUČIVANJE

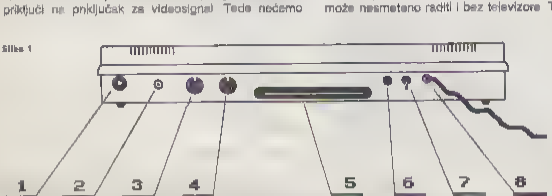
Da bi »orao« mogao raditi potrebno je da ga povežemo na mrežu preko mrežnog kabela koji se nalazi

na njegovoj poleđini. Osim spoje na mrežu potrebno je »orla« povezati s televizorom ili s videomonitorom. Za povezivanje služe predviđeni priključci (vidi sliku 1). Ako »orla« povežemo s TV prijemnikom potrebno je pravilno podesiti TV-prijemnik. To ćemo učiniti tako da ga prebacimo na »prvi program« i sliku potražimo oko devetog kanala. Potrebno je da na ekranu ugledamo malu bijelu zvjezdicu u gornjem lijevom kutu ekrana. Uz zvjezdicu treba da se nalazi trasperava crtica. Slika mora biti mirna i zvjezdicu moramo vidjeti dosta jasno. Ako smo uspjeli dobiti željenu sliku, »orao« je spreman za rad. Ako imate problema s pronalaženjem slike isključite ga. Ili je »orao« uopće uključen (prekidač mrežnog napona na poleđini vidi sliku 1) zatim da li ste pravilno povezali »orao« i televizor. Dosta je česte greške da se antenski kabel umjesto u antenski priključak u »orlu« priključi na priključak za videosignal. Tada nećemo dobiti sliku na televizoru. Ako sve previlno spojite dobit ćete na ekranu sliku koju je možda potrebno još malo »dotjerati«. To ćete postići potenciometrima za regulaciju svjetline i oštrine slike na televizoru. Potrebno je postići tamnu pozadinu (gotovo crnu) sa svjetlim slovima na kojima se jasno razaznaju točkice od kojih su slove sastavljene.

Previlna podešenost slike je jako važna za rad jer mutna slika ili slika koja podrhtava opterećuje oči pa rad pri takvoj slici nije zdrav. Može vam se desiti da poslijedni red slika izlazi izvan granica ekrana. To, da kako onemogućuje rad pa je potrebno na potenciometrime za regulaciju visine slike (ti se u previlu nalaze na poleđini televizora) podesiti visinu slike.

Ne kraju da budemo pošteni moramo reći da »orao« može nesmetano raditi i bez televizore. Televizor je tu

Slika 1



1 visina
2 TV ili jarmik
3 kanalofan
4 linija reč
5 konektor za priključenje
6 RESET tipka
7 ak tipka
8 kabel mrežnog napona

zbog toga da vidimo rezultate programa koje »orao« izvršava i bez njege ne možemo komunicirat s »orlom«.

Osim televizora na »orao« možemo priključiti i kasetofon u koji ćemo pohranjivati programe i štampač na kojem ćemo ispisivati rezultate našeg rada. O tome ćemo više u poglavljima koja se bave radom sa kasetofonom.

Bolji poznavatelji elektronike mogu preko vanjskog konektora povezati i neke druge sklopove na mikroračunalo »orao« (na primjer relaji za kontrolu nekih procesa).

KAKO POČETI

Kada smo obavili sve potrebne i pravilne priključili mikroračunalo »orao« dobili smo zvjezdicu iza koje se nalazi treptava crtica. Ova crtica naziva se kursor (cursor). Kursor nam pokazuje gdje će se ispisati slijedeći znak koji ukucamo s tastature. Svaki put kad ugledamo ovu bljeskavu crticu računalo očekuje da otipkamo neku naredbu ili podatak.

Zvjezdica koja se nalazi ispred kursora govori nam da se »orao« nalazi u monitoru (više o monitoru u posebnom poglavlju). Kako želimo raditi u BASICU, potrebno je kontrolu računala prebaciti u BASIC. Da to jednostavnije kažemo potrebno je »uključiti« BASIC. To ćemo postići tako da otipkamo BC. Na ekranu ćemo ugledati

```
*BC
```

Nakon toga pritisnete tipku (CR). Na ekranu će se pojaviti

```
>EAGLE< EXTENDED BASIC

    V 1.0 (C) 85

MEM SIZE  ?
```

Pritisnite tipku (CR) još jednom.

Sada će »orao« ispisati slovima memorijski prostor. Na ekranu će pisati

```
23534 BYTES FREE
ili
7158 BYTES FREE
```

To ovisi o veličini memorija koja je ugrađena u vašeg »orla«. »Orao« se pojavljuje u dvije varijante: sa 32 K ili sa 16 K memorije. U ovisnosti koju varijantu mikroračunala imate dobit ćete i komentar o raspoloživoj memoriji. Ispod obavijesti o slobodnoj memoriji vidimo blještavu crticu. Ona se naziva kursor i označava mjesto na koje će se upisati tekst koji otipkamo na tastaturi.

Ovaj znak pojavit će se svaki put kada računalo čeka da unesemo neku naredbu ili programsku liniju. Kada računalo izvodi program pojava ovog znaka signalizirat će nam da je program izveden ili da računalo od nas očekuje nekakav podatak ili da je nastupio prekid u izvođenju programa.

Prilikom unošenja programa ili nekih podataka, često skrećemo pogled s ekrana i radi toga kursor trepće. Tako ga lakše možemo uočiti pri pogledu na ekran.

Prije nego nastavimo s analizom rada s računalom treba napomenuti da nije moguće oštetiti (pokvariti) računalo tipkanjem na tastaturi. Bez obzira na to što ukucali uvijek možemo isključiti računalo i potom ga ponovno uključiti i početi od početka. Zbog toga ma žate slobodno iskušati sve što vam padne na pamet bez ikakva straha.

TASTATURA

Pogledajmo malo tastaturu mikroračunala »orao«. Oni koji su već sjedili za tastaturom pisaćeg stroja primijetit će veliku sličnost između ove dvije tastature. Ne lijevoj strani nalazi se velika skupina tipki na kojima su sva slova naše abecede uključujući i slova Q W X Y. Raspored je u skladu s jugoslavenskim standardom za raspored slova i nazivamo ga QWERTZ prema slovima u drugom redu tastature. Osim slova na tastaturi vidite i brojeve koji su smješteni u prvom redu. Vidimo da među brojevima postoje i brojevi ! ! @ kojih

na pisaćim strojevima nema. U programu na računalu nije dozvoljeno pisati malo slovo l umjesto broja 1 odnosno veliko slovo O umjesto broja 0. Da bi se nule lakše razlikovale od slova O preko nje je povučena kosa crta pa podsjeća na grčko slovo fi

Na nekim tipkama upisane su po dva znaka. Donje znakove dobivamo jednostavnim pritiskom na tipku. Da biste napisali gornje znakove morate os m tipke pritisnuti i tipku SHIFT (to su one dvije tipke na kojima ništa ne piše). Svejedno je koju od tipki SHIFT pritisnete ali je potrebno pritisnuti prije tipke na kojoj je znak koji želimo napisati i držati je sve dok se znak ne ispiše. Tipka SHIFT ima još jednu funkciju. Kad radite s uključenim malim slovima, pritiskom na ovu tipku i bilo koje slovo dobri ćete veliko slovo

Osim razmaknice (široka tipka bez oznake) koja služi za pisanje razmaka između riječi u ovoj skupini tipki vidite još i dvije tipke označene slovima CR odnosno CTL. T pka (CR) služi da računalu javite da ste završili unošenje podataka ili naredbe. Nju morate pritisnuti svaki put kad unesete bilo kakve naredbe. Tek kad je pritisnete računalo će razmotriti ono što ste napisali i ako je moguće izvršiti vaš nalog

Tipka (CTL) služi za specijalne funkcije i upotrebljava se u paru s nekim slovom slično kao i SHIFT. Ovdje vidite popis svih kontrolnih funkcija

(CTL) L briše sve s ekrana
(CTL) G zvučni signal
(CTL) F briše tekst od kursora do kraja ekrana
(CTL) H pomiče kursor ulijevo
(CTL) K pomiče kursor prema gore
(CTL) I pomiče kursor udesno
(CTL) J pomiče kursor prema dolje
(CTL) V uključuje zvučni signal prilikom tipkanja
(CTL) B briše tekst od kursora do kraja linije
(CTL) B uključuje štampač
(CTL) U isključuje štampač
(CTL) D vraća kursor na početak ekrana
(CTL) M vraća kursor na početak reda
(CTL) C prekida izvođenje programa

Od svih ovih funkcija najčešće ćete upotrebljavat (CTL) C kojom prekidamo izvođenje programe

Ne desnoj strani računala vidite dvije skupine po četiri tipke. Gornju skupinu tvore tipke označene strelicama koje služe za pomicanje kursora u sve četiri smjera. To vam je potrebno prilikom editiranja o čemu će još biti govore

Donje četiri tipke PF1 PF2 PF3 i PF4 su funkcijske tipke. Značenje ovih tipki je ovo

(PF1) prebacuje iz malih slova u velika i obrnuto
(PF2) služi za uključivanje štampača
(PF3) uključuje odnosno isključuje inverzni ispis slova
(PF4) kopira tekst ispod kursora

Tipke (PF4) podrobnije je objašnjena u poglavlju u kojem se govori o editiranju.

Tipkama na tastaturi pridružena je tzv. autorepeat-funkcija. To znači ako neku tipku malo duže držite počet će automatsko ispisivanje istog znaka

NEKI »STRUČNI« POJMOVI

U ovom poglavlju obješnjeni su neki od termina koja ćemo češće sretati u nastavku teksta.

RAM – To je kratica od Random Access Memory što možemo prevesti kao memorija sa slobodn-m pri stupom U ovu memoriju upisujemo svoj program u njoj se nalazi slika u nju računalo sprema međurezultate itd Sadržaj RAM-a gubi se nakon što se računalo isključi

ROM – Kratica od Read Only Memory što znači memorija koju možemo samo čitati Unutar ove memorije nalazi se program koji omogućuje računalu da razumije naše programe čita tastaturu crta sliku itd Sadržaj ove memorije upisan je u tvornici prilikom izrade računala i ne možemo ga mijenjati (zato i kažemo da je to memorija samo za čitanje) Sadržaj ROM-a ne gubi se nakon što isključimo računalo

K – Vjerojatno ste već primjetili da kad govorimo o veličini memorije iza broja stavljamo veliko slovo K. Kažemo na primjer da »neko« sadrževe 16K ROM-a. Slovo K mnogi čitaju kao »kilo« all to nije točno jer prefiks »kilo« znači tisuću (npr. kilometar je tisuću metara) Kad govorimo o računalima K označavе 1024 odnosno dve na desetu To znači da kad kažemo 16K ROM-a mislimo zapravo na 16 ×1024 odnosno 16384 bajta ROM-a U cijelom daljnjem tekstu oznaku K označavе 1024 bajta

BIT – Najmanja informacija kojom barata računalo Riječ bit kratica je od Binary digiT što znači binarne znamenke. Bit može imati vrijednost jedan ili nula

BYTE – (u daljnjem tekstu bajt) To je osnovna jedinica kompjutorske memorije Bajt je skup od osam bitova (jedinica ili nula) U njemu može bit vrijednost od 0 do 255 odnosno od nula do dve na osmu minus jedan Kada kažemo da računalo ima 16K bajta memorije to znači da može zapamtiti 16384 brojeva u rasponu od 0 do 255 Da pojednostavimo možemo reći da je svaki bajt jedno slovo odnosno da računalo može zapamtiti 15384 slove što odgovara tekstu od 9 stranica

ŠTO JE BASIC

Već sam u nekoliko navrata rekao da mikroračunalo »oreo« radi u BASICU i da ću u ovoj knjizi najviše govoriti o BASICU. Pogledajmo zato što je BASIC.

Ljudi se u međusobnom komuniciranju služe govorom. Pri tome postoje određeni jezici (na primjer hrvatski ili engleski jezik). Svi ljudi koji poznaju određeni jezik međusobno se razumiju. Ako nekome tko poznaje samo jedan jezik kažemo nešto na nekom drugom jeziku on nas neće razumjeti.

Kao i ljudi i računala razumiju jedan jezik. Jedan jezik ne računalu obuhvaća skup riječi koje tvore taj jezik. Jedan od takvih jezika je i BASIC. Kada kažemo da neko računalo »razumije« BASIC to znači da to računalo razumije skup riječi koje zajedno sačinjavaju jezik BASIC. Da bismo ove jezike razlikovali od jezika kojima govore ljudi, nazivamo ih programski jezici.

Među ljudima koji govore određenim jezikom ima manjih ili većih razlika između riječi koje oni upotrebljavaju, u ovisnosti o kraju iz kojeg potječu. Takve podskupove jezika nazivamo dijalektima.

Sva računala jednog tipa razumiju isti BASIC pa možemo program napisan na jednom »oreu« prenijeti na drugi »oreo«. Ne računalima različitih proizvođača postoje manje ili veće razlike između programskih jezika. Tako program napisan u BASICU na mikroračunalu »spectrum« neće raditi na mikroračunalu »oreo« sve dok se ne izvedu potrebne izmjene u programu. Zbog toga govorimo o dijalektima programskih jezika. Ova knjiga bav se dijalektom programskog jezika BASIC za mikroračunalo »oreo«.

Za razliku od jezika u svakodnevnom govoru za čije je učenje potrebno višegodišnji rad programske jezike možemo naučiti u vrlo kratkom vremenu. To slijedi iz toga što programski jezici sadržavaju vrlo malen skup riječi (otprilike). Kada nauč~mo jedan dijalekt nekog programskog jezika, potrebno nam je samo nekoliko sati da svladamo drugi dijalekt istog programskog jezika. Isto tako, ako dobro poznajemo jedan programski jezik, vrijeme potrebno za učenje svakog sljedećeg bit će sve kraće i kraće.

Programski jezik BASIC razvijen je u Sjedinjenim Američkim Državama prije dvadesetak godina. Prilikom njegova stvaranja osnovna ideja bile je da se stvori jezik koji će biti namijenjen. Zbog toga je ovaj program

ski jezik jednostavan za učenje i lako primjenjiv ne bilo koju vrstu problema

»Orao« kao ni sve ostala mikroračunale ne razumije naredbe BASICA već ih mora »prevoditi« u instrukcije strojnog jezika. To »prevođenje« je prilično spor proces, zbog toga su programi pisani u BASICU znatno sporiji od programa pisanih u strojnom jeziku. Ne dajte da vas riječi »sporije« dovedu u zabludu, jer računalo će i u BASICU mnoga zadatke obavljati jako brzo.

Preme načinu prevođenja razlikujemo dvije vrste rada s BASICOM. Prva je interpretirenje odnosno prevođenje svake instrukcije neposredno prije nego što se izvrši. BASIC s koji lako rade nazivamo interpreten Druga vrsta je kompajliranje odnosno prevođenje cijelog programa prije nego što počne njegovo izvršavanje. Takvi BASIC-i nazivaju se kompajleri. Kompajleri daju programe koji se znatno brže izvršavaju ali ipak još uvijek sporije od programa pisanih u strojnom jeziku. Međutim rad s interpreterima je znatno lakši jer je moguće lakše pronalaženje i ispravljanje grešaka i unošenje izmjene u program. (Kompajlirani programi praktično se ne mogu mijenjati.) Zbog toga je rad sa interpreterom jedini pogodan za početnike. Mikroračunalo »orao« posjeduje interpreter BASICA.

Rekao sam već da je BASIC razvijen u SAD (kao i većina drugih programskih jezika). Radi toga su sve riječi koje postoje u BASICU uzete iz engleskog jezika. Poznavanje engleskog jezika nije preduvjet za rad u BASICU jer je dovoljno znati šta koja komanda radi. Oni koji uče engleski jezik svakako će lakše zapamtiti pojedine komande ne temelju njihova značenja. U tabeli 1 nalaze se sve BASIC naredbe. O svakoj od njih bit će kasnije više govora.

PRIMJENA BASICA

U prethodnom poglavlju rekli smo da je BASIC relativno spor jezik. To treba shvatiti uvjetno jer ja »orlu« potrebno manje od tri sekunde da broji od 1 do 1000. U krugovima »stručnjaka« za mikroračunale vrlo često čuti da je basic loš spor nestrukturiran itd. Svi oni imaju neki svoj »najbolji« jezik i svoje »najbolje« računalo. Ne obazirite se ne te primjedbe jer ti isti »stručnjaci« (ako nešto znalu) učili su osnove programiranja baš u BASICU.

U BASICU su napisani mnogi ozbiljni programi. Na primjer tekst procesori (programi za obradu teksta)

programi za unakrsnu analizu brojeva, programi za crtanje itd. Ima i doste igara koje su napisane u BASICU, a mnoge od njih postigle su zavidan komercijalni uspjeh.

OVO JE GLUPOST

Kada pritisnete (CR) računalo će razmotriti što ste napisali i kako je to za njega zaista glupost; ispod toga će napisati

? SN ERROR

_

Time vas je »orao« obavijestio da ne razumije što ste napisali, e zatim ponovno postavio kursor spreman da izvrši vaše naredno naređenje

PA, DA POČNEMO!

Nakon što ste uključili računalo i ušli u BASIC, »orao« od vas očekuje naređenje što da radi. Ako otkucate bilo što i pritisnete (CR) računalo će razmotriti što ste mu naredili i ako je moguće, to će i izvršiti. Na ekranu treba da imate kursor (_) na početku reda. Ako to nije tako pođnite od početka. Pritisnite tipku na poleđini računala i ponovite postupak pozivanja BASICA. Sada napišite:

PRINT 1

Računalo će u narednom radu ispisati 1, a zatim ponovno kursor. Riječ PRINT je prva BASIC-naredba koju smo upoznali. PRINT na engleskom znači štampaj, a to isto znači i ova BASIC naredba. Kada napišemo PRINT 1 naređujemo računalu da napiše broj 1. »Orao« to razumije i zato odmah izvršava naše naredbe i ispisuje broj 1 u narednom radu

Napišite sada

```
PRINT 3+4
```

Nakon što pritisnete (CR) »orao« će u narednom redu napisati 7. Vidimo da nije napisao 3 + 4 već je izračunao izraz i napisao rezultat. Pod pojmom izraz podrazumijevamo bilo koji skup brojeva povezanih matematičkim operatorima. Možete napisati ove naredbe

```
PRINT 7-2
PRINT 8/4
PRINT 2*3
```

Svaki od ovih izraza »orao« će izračunati i ispisati vam rezultat. »Orao« i razpoznaje ove matematičke operatore

+ za zbrajanje
– za oduzimanje
* za množenje
/ za dijeljenje

Vidimo da se za množenje upotrebljava zvjezdica, a ne znak (x) niti točke kako ste možda navikli. Također nije dopušteno upotrijebiti dvije točke (:) kao znak za dijeljenje.

Napišite sada PRINT 3/2 i »orao« će napisati rezultat 1,5. Vidimo da umjesto zareza u decimalnim brojevima služi decimalna točka. To morate zapamtiti jer će »orao« ako napišete 7,3 to shvatiti kao dva broja (sedam i tri) a ne kao decimalni broj.

»Orao« pazi na prioritet matematičkih operacija. To znači da će on prvo izvesti matematičke operacije višeg prioriteta (množenje i dijeljenje) a tek potom operacije nižeg prioriteta (zbrajanje i oduzimanje). Napišite PRINT 3 + 2 * 4 i »orao« će najprije pomnožiti 2 i 4 a potom pribrojiti 3. I ispisati rezultat 11. Ako želimo promijeniti redoslijed izvršavanja operacija možemo upotrijebiti zagrade. Napišite

```
PRINT (3+2)*4
```

na ekranu će se ispisati rezultat 20. Dozvoljeni su i složenji izrazi, na primjer

```
PRINT (3+2)*((4-3)/(7-1)+1)
```

Kada trebamo više ugniježđenih zagrada (jedne unutar drugih) uzimamo uvijek okrugle zagrade a ne uglaste ili vitičaste kako ste učili u matematici. Osim matematičkih operacija zbrajanja, množenja, dijeljenja i oduzimanja postoji i operater za potenciranje. To je strelica prema gore. (Krenje desne tipke u drugom redu tastature a ne tipka iz skupine tipki sa strelicama). Kada pritisnete ovu tipku na ekranu će se ispisati okrenuto malo slovo v. Potenciranje ima najviši prioritet i izvodi se prije svih ostalih operacija osim ako zagrade ne određuju drugačije.

Sve što sam do sada opisao može i svaki prosječan kalkulator i to bez stalnog upisivanja naredbe PRINT. Pogledajmo sada nešto što »oraо« može a kalkulator ne može
Napišite

```
A=3
PRINT A
```

Na ekranu će se ispisati broj 3. Znak jednako (=) ima ne mikroračunalima posebno značenje pridruživanja vrijednosti. Kada napišete A=3 u memoriji računala pamti se broj 3 kojemu je pridruženo ime A. Ovako spremljen podatak nazivamo varijabla.

Varijable je prostor u memoriji kojem je pridijeljeno neko ime i vrijednost. Ime varijable sastoji se od dva znaka od kojih prvi mora biti slovo engleske abecede (nisu dozvoljene naša slova č, đ, ć, ž) a drugi znak može biti ili slovo engleske abecede ili broj. Kao što ste vidjeli u primjeru ime varijable može biti i samo jedno slovo.

Nakon što varijablu jednom definiramo možemo se s njom poslužiti i u izrazima. Možete napisati

```
PRINT A+2
```

i »oraо« će napisati rezultat 5 (ako ste prethodno definirali A=3). Vrijednost pojedine varijable može se neograničeno puta mijenjati (otud i njeno ime) prema tome možete napisati

```
A=5
PRINT A
A=7
PRINT A
```

i vidimo da je naredba PRINT A u prvom primjeru ispisala rezultat 5 a u drugom 7. Ako neku varijablu nismo definirali, a zahtijevamo od računala da nam ispiše njenu vrijednost, dobit ćemo rezultat 0. Možemo reći da sve varijable prije nego što su definirane imaju vrijednost nula.

Vrijednost varijable može se i izračunavati na primjer

```
X = 7+4
PRINT X
```

Ovdje se najprije izračunava vrijednost izraza s desne strane a zatim se dobivena vrijednost pridružuje varijabli X. Moguć je i izraz

```
A=5
X=A*2
PRINT X
```

Sada smo u definiciji varijable X upotrijebili prethodno definiranu varijablu A. Pri tome se treba držati pravila da se s lijeve strane znaka jednakosti nalazi samo varijabla koju definiramo, a s desne strane se mogu nalaziti bilo kako složeni izrazi u kojima imamo jednu ili više varijabli. Potrebno je jedino da su sve varijable koje upotrebljavamo s desne strane znaka jednakosti već prethodno definirane.

Jedan od klasičnih primjera primjene varijabli je definiranje broj PI. Ako morate izračunavati više izraza u kojima se spominje PI, napišite

```
PI=3.141
```

Nakon toga možete površinu kruga koji ima polumjer 5 izračunati ovako

```
PRINT 5*5*PI
```

Možete to napisati i ovako

```
R=5
P=R*R*PI
PRINT P
```

Vjerujem da ste primijetili da se u prethodnom primjeru kvadrat broja R računa izrazom R*R. Umjesto toga mogli ste napisati potenciranje i eksponent 2, ali je prvi postupak brži i točniji.

PISANJE TEKSTA

Osim rada s brojevima, »oraz« može raditi i sa slovima. Za pisanje slova služi nam naredba PRINT kao i za brojeve. Da bi »oraz« ispisao neki tekst, on mora biti u navodnicima. Upišite ovu liniju

```
PRINT "ZDRAVO, JA SAM ORAZ."
```

Na ekranu ćete ugledati ispis teksta koji ste naveli u navodnicima. Takav tekst u kompjuterskoj terminologiji naziva se string. Unutar navodnika može se nalaziti

bilo kakav tekst kombiniran od svih slova i svih raspoloživih znakove izuzevši znakove navoda. To je zato što će »orao« kad naiđe na znak navoda unutar stringa smatrati da je tekst završen.

Prilikom pisanje teksta jedno je ograničenje da ukupna dužina stringa ne smije biti veća od 255 znakova. Pri tome se broje svi upotrijebljeni znakovi, uključujući i razmake.

Kao što postoje varijable za »spremanje« brojeva postoje i string-varijable u koje smještamo tekst. One imaju ime koje se sastoji od jednog ili dva slova (kao i numeričke varijable) i znaka ($). Ovaj znak je obavezan da bi računalo znalo da je riječ o string varijabli. Možete napisati

```
A$="DOBAR DAN"
PRINT A$
```

i računalo će napisati DOBAR DAN. Svaki put kada napišemo PRINT A$ računalo će ispisati tekst koji smo pridružili string-varijabli A$.

Pri definiranju string varijable možemo upotrijebiti već prije definirane string-varijable. Možete napisati

```
B$="ZDRAVO"
C$=B$
PRINT C$
```

i računalo će napisati ZDRAVO. Moguća je čak i zbrajanje string-varijabli. Dopušten je izraz

```
A$="DOBAR"
B$=" DAN"
C$=A$+B$
PRINT C$
```

Na ekranu će se pojaviti tekst DOBAR DAN. Zbrajanje stringova je zapravo nadovezivanje više stringova jedan na drugi. Kao s numeričkim varijablama dozvoljeno je kombiniranje string-varijabli i stringova. Vidimo to u primjeru

```
A$="JA SAM "
B$=A$+"ORAO"
PRINT B$
```

Ako neku string varijablu niste definirali a naredite računalu da je ispiše, računalo neće ispisati ništa. String varijable koje niste definirali imaju dužinu nula. Njih zovemo prazan string. Ako želite neku string-varijablu »obrisati« možete napisati

```
A$=""
```

Kao što smo već rekli «orao» razumije samo određeni set naredbi BASICA. Naredbe moramo nep sati doslovce onako kako se pišu. Nije dozvoljeno ispuštati pojedina slova, kratiti naredbe niti unutar naredbe stavljeti razmake. Između naredbe i argumenta obično stavljamo razmak, iako se može i izostaviti. Na primjer...ovi su izrazi za «orla» potpuno jednaki

```
PRINT 1
PRINT1
```

U drugom primjeru vidimo da između riječi PRINT i broja 1 nema razmaka. To nikako nije dobro jer pravilno upisane naredbe s razmacima između njih pridonose čitljivosti programa a to je osobito za početnike izuzetno važno. Jedini primjer kada s razmacima moramo zaista biti oprezni jest pisanje tekstova unutar navodnika. Sve ono što piše unutar navodnika «orao» će uzeti točno u onom obliku kako je napisano

KOMBINACIJE IZRAZA U PRINT-NAREDBAMA

Upišite ovaj primjer

```
PI=3.141
R=5
P=R*R*PI
A$="POVRŠINA JE "
PRINT A$
PRINT P
```

Dobiveni ispis nije jako pregledan a između teksta POVRŠINA JE i ispisanog rezultata nalazi se tekst naredbe PRINT P. Upišite sada prethodni primjer ponovno jedino umjesto posljednje dvije PRINT naredbe upišite naredbu

```
PRINT A$;P
```

Sada je ispis mnogo ljepši. Tekst POVRŠINA JE i rezultat ispisani su u jednoj liniji. Vidimo da unutar jedne PRINT-naredbe može biti više izraza koji su međusobno odijeljeni točka zarezom. Točka zarez nalog je mikroračunalu da oba izraza ispiše u istom redu i da između njih ostavi mali ili nikakav razmak

Mali razmak ostavit će kada ispisuje dve broja jeden iza drugog (tada je razmak dve prazna mjesta) kada ispisuje tekst iza broja ili broj iza teksta (tada je razmak jedno prazno mjesto)

Kada ispisuje dve tekstа jedan iza drugoga, neće ostaviti nikakav razmak. To nam omogućuje da tekstove nastavljamo jedan za drugim bez potrebe da ih združate.

Unutar jedne PRINT-naredbe može se nalaziti po volji mnogo izraza koji su međusobno odijeljeni točka zarezom. Jedino ograničenje je da ukupna dužina linije ne može biti veća od 72 znaka.

Osim točka zareza za odjeljivanje tekstova možemo uzeti zarez. Kada »orao« nađe na zarez, on će spuštanje nastaviti od desetog znaka u redu a ako je je prošao od dvadesetog i tako dalje. Ako je prošao i trideseti znak prijeći će u novi red. Pozicije na koje će ići ispis zovu se tab-pozicije (zato što nam služe pri ispisu tablica). Možemo naći da neiskom ne zarez »orao« skače na prvu slobodnu tab-poziciju.

Ako između dva izraza nap šemo dva ili više zareza ispis će se pomaknuti za onoliko tab pozicija koliko smo zareza upisali. Ako izreze odvajamo točka zarezom onda će jedna točka zarez imati potpuno isti efekt kao i više njih.

U jednoj PRINT-naredbi za odjeljivanje izraza možemo upotrebljavati i točke zareze i zareze, već prema tome što nam ne iom mjestu odgovara.

MOJ PRVI PROGRAM

Sve što smo do »sada unosili u računalo izvršavalo se odmah nakon što smo ukucali. Napišite sada narednu

```
10 PRINT 1
```

Kao što vidite, računalo nije izvršilo naredbu PRINT a nije ni prijavilo grešku. Ako sada upišete

```
LIST
```

računalo će ponovno ispisati liniju 10 PRINT 1 (ako vam ispiše još nešto pritisnite tipku na poleđini računala ponovno pozovite BASIC i unesite naredbu 10 PRINT 1)

Vidimo da je računalo zapamtilo liniju 10 PRINT 1. Takve linija nezi va se programskom linijom i za razliku od komendna linija počinje brojem i ne izvršava se sve dok to ne naredimo. Pomoću ovakvih programskih linija

pišemo programe. Komanda LIST nalog je računalu da ispiše sve programske linije (cijeli program) koje ima u memoriji

Kad unesimo određenu komandu računalo odmah razmotri što smo napisali i ako smo pogriješili odmah nas o tome obavijesti tekstom ? SN ERROR Kad unesimo programsku liniju tada računalo prihvati svaki tekst koji počinje brojem e na provjerave da li je tekst ispravan Tekst će računalo provjeriti tek kada bude izvršavalo tu programsku lin ju

Napišite sada

```
RUN
```

Na ekranu će se pojaviti broj 1 Taj broj napisan je kao rezultat izvršavenja programske linije broj 10 Komanda RUN nalog je računalu da izvrši program koji se nalazi u memoriji (RUN na engleskom znači trčati)

Naredbu PRINT mogli smo upotrijebiti kao komandu, pa se onda odmah izvršavala Ili je staviti u programsku liniju Naredbe LIST i RUN nećemo stavljati u programske linije već će nam služiti jedino kao komande Kasnije ćete vidjeti da ime I nekih koje služe isključivo u programskim linijama

Kada smo unijeli neku programsku liniju možemo je lagano izbrisati tako da otkucamo samo njen broj i nakon toga pritisnemo (CR) Sve programske linije odnosno cijeli program možemo izbrisati naredbom

```
NEW
```

Ova naredba obrisat će sve programske linije i sve varijable i string-varijable koje se nalaze u memoriji NEW na engleskom znači nov odnosno nakon te naredbe najčešće upisujemo novi program

Vratimo se sada našem primjeru izračunavanja po vršine kruga Najprije ćemo obrisati sve naredbom NEW a zatim upisati ovaj tekst

```
10 R=5
20 PI=3.141
30 P=R*R*PI
40 PRINT "POVRŠINA JE ";P
```

Nakon što ste unijeli ovaj tekst obrišite ekran pritiskom ne tipke (CTL) L. Poslije toga naprište LIST Sada ne ekranu vidite ispisan vaš program To je prvi program koji ste unijeli u računalo Pažljivo ispitajte da li je sve dobro upisano i zatim pokrenite program naredbom RUN Ako ste pri provjeri programa otkrili neku neispravnu liniju Il' ako je niste primijetili e računalo je našlo ne nju (to će vam javiti porukom ? SN ERROR) možete neispravnu liniju popraviti tako da jednostavno otkucate ispravnu liniju s istim brojem Tada će računalo najprije obrisati staru liniju, a zatim upisati umjesto nje novu liniju.

Pogledajte brojeve linija u našem programu. Vidimo da ti brojevi rastu i da se povećavaju za 10. Svaka programske linije mora imati svoj broj. Ako ge nema izvršit će se odmah kao komande. Taj broj računalu služi kao oznaka da je u pitanju programska linija a i određuje redoslijed kojim linije idu. Gornji primjer možete upisati i odozdo prema gore računalo će ipak pravilno složiti program jer ga slaže po brojevima programskih linija.

Brojevi programskih linija povećavaju se za 10 zbog toga da između njih ostane mjesta za unošenje novih linija. Umjesto povećanja od 10 možete staviti i neku drugu vrijednost. Običaj je da se uvijek ostavlje razmak 10 između dvije linije jer je praksa pokazala da je to dovoljno.

Napišite sada (bez brisanja programa)

```
35 PRINT "PULUMJER JE ";R
```

Pokrenite ponovno program (RUN) i vidjet ćete da će »oraо« prvo napisati poluměje pa onda površinu. Izlistajte program (naredbe LIST) i vidjet ćete da je naredba 35 smještena između naredbi 30 i 40. To je to ubacivanje naredbi o kojemu sam govorio.

Važno je da uočimo kako se izvršavaju naredbe u programu. Računalo najprije izvrši prvu naredbu (naredbu s najmanjim brojem) potom drugu itd. To je tzv. tok izvršenja programa. Kada god vam se u programu dogodi greška, program analizirajte prateći tok njegove izvršenja. Tako ćete ubrzo pronaći grešku. Unesite sada ovaj tekst

```
25 GLUPOST
```

Pokrenite program sa RUN. Računalo će ispisati

```
?SN ERROR
 25 GLU?POST
```

To je one iste greške koju smo vidjeli i u prethodnim primjenma, jedino što je ispod nje ispisano linija 25. Ovim nam računalo jevlje da je u pitanju greške u liniji 25. Time nam olakšava otkrivanje i ispravljanje grešaka. Unlju 25 možemo obrisati tako da ukucamo broj 25 i pritisnemo (CR) i naš će program opet biti ispravan.

Greške SN s kojom se tako često sređamo zapravo je sintaktičke greška. To znači de emo nešto krivo napisali odnosno da tekst nije razumljiv računalu. Takve greške dešavaju se dosta često (namojte de vas to upleši sintaktičke greške prave i profesionalni programeri). Takve greške nisu opasne jer se vrlo lako uočavaju i brzo ispravljaju. To su tzv. formalne greske. Osim tih postoje još i logičke greške na primjer kada pridružimo krivu vrijednost nekoj varijabli. Uz ove greške program radi (računalo ne zna što smo mi željeli pa ne može otkriti grešku) ali rezultat nije ispravan.

i akve su greške mnogo teže za ispravljanje pa zato i opasnije.

Upišite sada u program koji imate u memoriji i naredbu

```
5 PRINT "RAČUNANJE POVRŠINE KRUGA"
```

Ako sada pokrenemo program dobit ćemo tekst

```
RAČUNANJE POVRŠINE KRUGA
POLUMJER JE  5
POVRŠINA JE  78.525
```

Pokrenite sada program ponovno ali umjesto naredbe RUN napišite RUN 10. Vidimo da sada tekst RAČUNANJE POVRŠINE KRUGA nije ispisan. To je zbog toga što je naredba RUN 10 naredile računalu da izvršavanje programa započne od programske linije broj 10. Tako možemo izvršavanje programe započeti od bilo koje linije. Napišite sada tekst

```
A=15
```

Znamo da smo tim varijabli A pridružili vrijednost 15. To možemo ispitati ako napišemo PRINT A. Pokrenite sada program sa RUN i nakon što se izvrši napišite ponovno PRINT A. Vidimo da smo sada dobili vrijednost 0 odnosno da je računalo zaboravilo vrijednost varijable A (reći smo da svaka nedefinirana varijabla ima vrijednost 0). To se dogodilo zbog upotrebe naredbe RUN. Ove naredbe prije nego pokrene program izbriše sve vrijednosti svih varijabli uključujući i string-varijabla.

Pokrenite sada program naredbom RUN 20. Vidimo da su sada polumjer i površina jednaki 0. Zašto se to dogodilo?

Pogledajmo sve korak po korak. Naredba RUN 20 prvo je obrisala sve varijable uključujući i varijablu R u kojoj se nalazi vrijednost polumjera. Nakon toga izvršavanje programa počelo je od linije 20. U toj liniji definirane je vrijednost varijable PI zatim je u naredbi 30 izračunane vrijednost varijable P. s pri tome je upotrijebljena varijabla R. Kako smo liniju 10 u kojoj se pridružuje vrijednost varijabli R preskočili u trenutku izračunavanja površine varijabla R imala je vrijednost nula. Vidimo da se prilkom pozivanja programa iz sredine mogu desiti greške pa zato s tekvim povezivanjem trebe biti oprezan.

Napišite sada RUN 11. Računalo će napisati

```
?US ERROR
```

Ovim nas »orso« obavještava de smo zahtijevali izvođenje linije koja ne postoji

Slično kao i u naredbi RUN i iza naredbe LIST može se nalaziti broj. Nap šite

```
LIST 30
```

Na ekranu vidite ispisanu liniju 30 odnosno naredbu LIST 30 nelog je računalu da izliste liniju 30. Ako u naredbi LIST navedete broj koji na postoji »oraz« neće ispisati ništa. Iza naredbe LIST možete navesti i dva broja odvojena crticom. Na primjer naredba LIST 20–35 ispisat će sve linije od 20 do 35 uključujuci i linije 20 i 35. Ako izostavite zadnji broj bit će izlistane sve i nije od linije koju ste naveli do kraje a ako izostavite prvi broj bit će izlistane sve linije od početka do linije koju ste naveli.
Probajte

```
LIST 10–
LIST –30
```

Ova mogućnost najviše će vam koristiti kada budete izlistaval duge programe koji ne stanu odjednom na ekran. Tada će nekon naredbe LIST bez navedenih granica cijeli program »prelatjet« ispred vaših očiju i vidjet ćete samo posljednjih trideset tinija.

ISPRAVLJANJE TEKSTA PROGRAMA (EDITOR)

Ponekad je potrebne prom jenti nešto u već napisanom programu. To činimo bilo zbog toga šio imamo griješku u programu ili zbog toga što želimo promijeniti neke vrijednosti u programu. Do sada smo se služili mogućnošću da umjesto linije s greskom upišemo is pravnu liniju. »Oraz« nam pruže mogućnost uredivenje programe na ekranu. To se na engleskom jeziku nazive screen editing pa zato kažemo da »orso« ime skrin editor.

Pri tom poslu služimo se tipkama za pomicanje kur sore (to su one četiri tipke sa strelicama) i tipkom copy (to je tipka sa oznakom PF4). Pogledajmo to na primje ru. Ukucajte doslovno ovaj program

```
10 A=5
20 PRINT "A= ";A
30 PRINT "A*A= ";A*A
40 PRINT "A^2= ";A/2
```

U ovom programu postoje dvije greške. Umjesto točke zareze u liniji 30 otkucane je dvotočka a umjesto znaka dijeljenja u liniji 40 otkucan je upitnik. Prve greške je formalna greška i nju će nam računalo prijaviti kad pokrenemo program. Pozabavimo se prvo ispravkom te greške.

Dakako, možemo umjesto linije 30 unijeti liniju 30 bez greške i tako popraviti grešku. Kada linija koju popravljamo nije duga, tada je to vrlo lagano učiniti. Međutim, ako imamo dužu liniju, tada je lakše liniju prepraviti. To ćemo mi učiniti u ovom primjeru. Postupak je ovaj

– obrišite ekran sa (CTL) L
– izlistajte program sa LIST
– uz pomoć strelica za pomicanje kursora gremo gore idite na liniju 30
– pritiskajte tipku (PF4) sve dok kursor ne dođe na dvotočku
– otkucajte točku zarez
– pritiskajte (PF4) sve dok kursor ne proda kraj linije (na prvo prazno mjesto iza linije)
– pritisnite (CR)
– obrišite ekran sa (CTL) L
– izlistajte program sa LIST

Vidimo da sad na ekranu imamo ispisan program sa ispravljenom greškom u liniji 30. Ako to nije tako provjerite ovo poglavlje od početka jer negdje ste pogriješili. Pazite da preko onih dijelova linije koje želite zadržati prelazite pomoću tipke (PF4) a ne pomoću strelice desno jer tipka (PF4) kopira znakove preko kojih prelazi.

Pokrenite tako ispravljen program. Vidjet ćete da računalo izvršava program bez obzira na grešku u liniji 40. To je zbog toga što je u liniji 40 greška logičkog tipa. Takst unutar navodnika računalo piše onako kako smo ga naveli i ne postoji način da sazna da smo mi umjesto znaka up.trik željeli napisati znak dijeljenja. Ispravite tu grešku potpuno istim postupkom koji ste primijenili u prvom primjeru.

Sada imamo program koji nema greške i koji se ispravno izvršava. Ako poželimo da program umjesto kvadrata broja 5 i polovice broja 5 ispisuje iste vrijednosti za broj 7 tada možemo liniju 10 prepraviti u 10 A= 7 primjenjujući isti postupak kao i u ispravljanju grešaka. Ubacite sada liniju 25 PRINT. Pokrenite program i vidjet ćete da program između ispisa vrijednosti broja i njegova kvadrata ostavlja jednu praznu liniju. Kada god to želimo učiniti možemo ubaciti naredbu PRINT bez ikakvih argumenata.

Ako želimo ubaciti i između naredbi 30 i 40 naredbu PRINT možemo kursorom otići na naredbu 25 otkucati broj 35 i pomoću tipke PF4 iskopirati ostatak linije. Time smo zapravo iskopirali naredbu 25 u naredbu 35

e naredba 25 je ostale neoštećena. To možete vidjeti ako izlistate program

Prilikom normalnog unošenja naredbi strelica ljevo nam služi za brisanje posljednjeg unesenog znaka. Na primjer ako greškom otkucamo PRINTT i nakon toga pritisnemo strelicu ljevo obrisat ćemo suvišno slovo T i ostat će pravilno napisana naredba PRINT.

Nakon što pritiskom na strelicu gore ili dolje otvorimo kursor za kopiranje više ne možemo pomoću strelice ljevo brisati kopirane znakove jer ona sad služi za pomicanje kursora. Posljednji uneseni znak možemo sada obrisati istovremenim pritiskom na tipke (CTL) i H.

PETLJA

Svi programi koje smo do sada upisali izvršavali su se od prve linije prema posljednjoj i nakon što je ova izvršena izvršavanje programa se prekinulo. Takav rad dovoljan je za jednostavne primjere kojima smo se do sada bavili ali zamislite da morate napisati sve brojeve od 1 do 10 000 i da upisujete

```
10 PRINT 1
20 PRINT 2 itd.
```

Osim što bi ovo bilo mukotrpno i strašno dugo trajalo takav program ne bi stao u memoriju mikroračunala.

Najveća prednost računala je njihova sposobnost da određene naredbe ponavljaju beskonačno ili konačno mnogo puta. To se postiže naredbama koje utječu na tok izvršavanja programa. Napišite program

```
10 PRINT "ZDRAVO ";
20 GOTO 10
```

Pokrenite ovaj program sa RUN. «Oraz» će početi ispisivati ZDRAVO i kad ispuni cijeli ekran pomicat će tekst prema gore i dolje dodavati novu liniju ispunjenu tekstom. Ovaj program izvrtit će se sve dok ga ne prekinemo. Prekid programa postići ćemo pritiskom na tipke (CTL) i C (morate ih zadržati nekoliko sekundi) «Oraz» će napisati

```
BREAK
10 PRINT "ZDRAVO";
>li
BREAK
20 GOTO 10
```

što ovisi o tome koju je liniju izvršavao u trenutku kada smo ga prekinuli. Preanalizirajmo malo ovaj program.

Linija 10 nalog je računalu da ispiše niz ZDRAVO. Točka zarez na kraju ove naredbe nalog je da se slijedeći tekst piše u istom redu s malim razmakom. Nakon što izvrši ovu liniju računalo nađe na liniju 20 u kojoj stoji naredba GOTO. Go to na engleskom znači – idi na. Broj iza naredbe GOTO govori računalu od koje naredbe će se nastaviti izvršavanje programa. U našem primjeru to je linija 10 pa računalo ponovno izvrši liniju 10. Nakon nje ponovno nađe na liniju 20 i sve se opet ponavlja. Kažemo da računalo izvršava petlju. S obzirom na to da se ova petlja izvršava sve dok je ne prekinemo, nazivamo je beskonačnom petljom. Pogledajte ovaj program

```
10 A=1
20 A=A+1
30 PRINT A
40 GOTO 20
```

Pokrenite program i vidjet ćete da isp suje rastući niz brojeva. Preanalizirajmo zajedno taj program. 60 END

Linija 10 postavlja u varijablu A vrijednost 1. Linija 20 uzima vrijednost varijable A, dodaje joj 1 i rezultat smješta nazad u varijablu A. (Ovdje vidimo da znak jednakosti ima funkciju pridruživanja jer izraz A = A + 1 nije matematički prihvatljiv). Ovu naredbu nazivamo još i brojač jer ju često upotrebljavamo u programima kada nam je potrebna brojanja. L nije 30 isp suje vrijednost varijabla A a linija 40 vraća zvođenje programa na liniju 20 koje ponovno povećava vrijednost varijabla A. Ponovno imamo beskonačnu petlju al se ovaj puta unutar nje mijenja vrijednost varijabla A. Dakako i ovaj ćemo program prakinuti uz pomoć tipki (CTL) i C

Naredbu GOTO možemo iskoristiti i za promjenu toka programa baz zatvaranja petlje. Preanalizirajte sam zašto se u slijedećem primjeru na ispisuje broj 2

```
10 PRINT 1
20 GOTO 40
30 PRINT 2
40 PRINT 3
```

Osim beskonačnih petlji postoje i petlje sa određenim brojem ponavljanja. Počnimo s jednostavnim primjerom

```
10 FOR A=1 TO 10
20 PRINT A
30 NEXT A
```

Pokrenite ovaj program i vidjet ćete da ispisuje brojeve od 1 do 10. Ovdje se sređenno sa jednim parom naredbi koje uvijek idu zajedno. Prve naredba je naredba FOR unutar koje se nalazi jedna varijabla. Naredba FOR u liniji 10 znači – mijenjaj A od 1 do 10. Neitaskom ne ovu naredbu računalo pridruži varijabli A vrijednost 1 i nastavi izvođenje. Nakon što ispiše varijablu A u liniji 20 računalo naide na liniju 30 u kojoj piše NEXT A. Ova naredba znači slijedeći A i nakon nje računalo će se vratiti na naredbu 10 i pridružiti varijabli A vrijednost 2. Nakon 10 krugova varijabla A imat će vrijednost 10. Kada računalo naide na naredbu NEXT A, skočit će na naredbu 10 povećati A za jeden i tada ustanoviti da je A sada veći od gornje granice navedene u naredbi FOR. Zato će se zvrsavanje programa nastaviti iza naredbe NEXT. Kako mi iza ove naredbe nemamo drugih naredbi izvođenje programa je gotovo.

Takve petlje sa unaprijed određenim brojem ponavljanja upotrebljavamo mnogo češće od beskonačnih petlji. U primjeru vidimo program koji ispisuje kvadrate brojeva od 5 do 15

```
10 B=5
20 G=15
30 FOR A=B TO G
40 PRINT "A= ";A,"A*A= ";A*A
50 NEXT A
60 PRINT "KRAJ"
```

U liniji 30 vidimo da za određivanje granica možamo uzeti i varijable a ne samo konstante. Isto tako treba uočiti da se nakon što se dosegne gornja granica izvršavanje programa nastavlje na liniji iepod linije koja sadrževa NEXT A, pa se zbog toga ne kraju ispisuje riječ KRAJ.

Ponekad nam je potrebno da se varijabla u petlji mijenja s korakom različitim od 1. Tu ćemo upotrijebiti naredbu STEP koja je dio naredbe FOR što se vidi iz primjera

```
10 FOR A=1 TO 100 STEP 7
20 PRINT A
30 NEXT A
```

Vidimo da se u ovom primjeru brojevi povećavaju s korakom 7 što tako vidimo da gornja granica (100) nije dostignuta. To u petlji FOR nije obavezno čim vrijednost varijable premaši gornju granicu izvršavanje petlje se prekida.

Step odnosno korak može biti i negativan ali tada je potrebno da je donja granica veća od gornja. To vidimo u primjeru

```
10 FOR A=5 TO 1 STEP -1
20 PRINT A
30 NEXT A
```

Važno je još jednom napomenuti da je varijable nakon izvršenja petlje veća od gornje granice petlje (odnosno manja ako je step bio negativan).

S obzirom na to de varijable koje je upotrijebljena u petlji FOR ima funkciju da kontrolira izvršenje petlje, ovakvu varijablu nazivamo kontrolna varijable.

Vidjeli smo da upotreba naredbe STEP u petlji FOR nije obavezna. Ako ne navedemo STEP podrazumijeva se STEP 1.

Treba biti oprezan ako se vrijednost kontrolne varijable mijenja unutar petlje. Tada broj izvršenja petlje neće biti jednak predviđenom broju izvršenja a ako kontrolnu varijablu unutar petlje smanjujemo može se čak desiti da nam se petlja FOR pretvori u beskonačnu petlju. »Omo« nam dopušta da u naredbi NEXT izostavimo ime kontrolne varijable. Ovo nemojte prakticirati jer ako navodite ime kontrolne varijable lakše ćete vidjeti na koju se naredbu FOR odnosi naredba NEXT što je nužno potrebno kada imamo više naredbi FOR unuter jednog programa.

Ako su gornja i donja granica u petlji FOR jednake ili je gornja granica manja a STEP pozitivan ili je gornja granica veća a STEP negativan petlje će se izvršiti samo jednom.

Unutar jedne petlje FOR može se nalaziti druga petlja FOR! To je ilustrirano u primjeru

```
10 FOR A=1 TO 10
20 FOR B=A TO A+3
30 PRINT B;
40 NEXT B
50 PRINT
60 NEXT A
```

U ovom primjeru naredba 50 PRINT nema funkciju razmicanje redove već jednostavno prebacuje ispis u novi red s obzirom na to da je prethodni PRINT završio sa točka zarezom. Obrišite naredbu 50 pa pogledajte što se dešava.

Kada unutar neke petlje postevite drugu petlju kraj (naredbe NEXT) MORA također biti unutar petlje. Odnosno kažemo da petlja može biti unutar petlje al se ne smiju preklapati. U protivnom, program neće raditi

U prethodnom primjeru vidimo u naredbi 20 da je u unutrašnjoj petlji pri definiranju granice dozvoljeno upotrebljavati kontrolnu varijablu iz vanjske petlje te da je umjesto broja ili varijable kao granicu dopušteno navesti i izraz. To vrijedi za većinu slučajeva. Gotovo uvijek kada računalo očekuje broj možemo umjesto njega navesti varijablu ili matematički izraz.

Napisat ćemo program koji računa površinu pravokutnika. Za vrijeme rada programa unijet ćemo stranice A i B a program će ispisati površinu

```
10 INPUT A
20 INPUT B
30 P=A*B
40 PRINT "POVRŠINA JE ";P
50 GOTO 10
```

Kad pokrenemo ovaj program na ekranu će se pojaviti znak upitnika (?). To je znak da je računalo naišlo na instrukciju INPUT i da očekuje da mu upišemo neki broj. Otkucat ćemo broj (na primjer 3) i pritisnuti tipku (CR). Na ekranu će se pojaviti novi upitnik. Računalo je naišlo na drugu INPUT-naredbu i očekuje od nas da mu damo novu vrijednost. Kada i ovdje unesemo vrijednost (na primjer 4) i pritisnemo (CR) računalo će nam ispisati POVRŠINA JE 12. Nakon toga pojavit će se ponovno upitnik što je znak da je računalo naišlo na liniju 50 koja se je poslala na ponovno izvršavanje programe od linije 10. Vidimo da i ovdje imamo beskonačnu petlju. Program će se izvršiti sve dok ga ne prekinemo. Kad na ekranu piše upitnik a želimo prekinuti program potrebno je da pritisnemo tipke (CTL) i C a zatim tipku (CR).

UNOS PODATAKA (INPUT)

Svi programi koje smo do sada izrađivali definirali su vrijednosti varijabli koje se u programu upotrebljavaju tako da je svako izvođenje programe uvijek davalo isti rezultat. Ako bismo u takvom programu poželjeli promijeniti vrijednost neke varijable morali smo unijeti izmjene u sam program. Vrlo često nam je potrebno da neke brojeve unosimo za vrijeme rada programa.

Proučimo još malo što su učinile naredbe INPUT. Napišite PRINT A. Računalo će ispisati broj 3. Vidimo

da varijable A ima vrijednost 3. Tu vrijednost pridružili smo varijabli A kada je računalo izvršavalo naredbu INPUT u liniji 10. Dakle naredba INPUT zaustavlja izvršenje programa i zahtijeva od nas da unesemo podatak koji se pridružuje varijabli navedenoj u naredbi INPUT.

Pokrenite ponovno prethodni program i kada računalo napiše upitnik otipkajte neku slovu umjesto broja. Računalo neće razumjeti to što smo mu napisali jer očekuje brojnu vrijednost. Zbog toga će napisati

```
?REDO FROM START
```

Time nam računalo javlja da nije razumjelo to što smo upisali i zahtijeva od nas da upišemo pravilnu vrijednost koja će se onda pridružiti varijabli navedenoj u naredbi INPUT. Vidimo da prilkom izvršenje naredbe INPUT nije dopušteno uniješi varijabiu ili izraz umjesto brojne vrijednosti.

Napišite sada INPUT A kao komandu (bez broja naredbe). Vidite da je računalo prijavilo grešku. Naredba INPUT može se nalaziti uključivo u BASIC-liniji i nije ju dozvoljeno navoditi kao komandu.

U našem primjeru imali smo unošenje dvije varijable i za svaku od njih upotrijebili smo posebnu naredbu INPUT. Neš prethodni program možemo napisati i ovako

```
1Ø INPUT A,B
2Ø P=A*B
3Ø PRINT "POVRŠINA JE ";P
4Ø GOTO 1Ø
```

Kada sad pokrenemo program računalo će ispisati upitnik, ali računalo sada očekuje da mu damo dva broja. Možemo zbog toga napisati 3 4 (reći smo da decimalne brojeva pišemo s točkom ne primjer 3 4 e da zarez služi za odvajanje brojeva) Kada upišemo 3 4 računalo će uzeti prvi broj (broj 3) i pridružiti ga prvoj varijabli navedenoj u naredbi INPUT (varijabli A) Nakon toga će uzeti drug broj i pridružiti ga drugoj varijabli navedenoj u naredbi INPUT (varijabli B) Posilje toga program ce se normalno izvršavati kao i u prvom primjeru

Ako u ovom programu kada se pojavi upitnik upišamo samo jedan broj računalo će uzeti taj prvi broj pridružiti ga varijabli A, a zatim ce napisati u sljedećem radu dve znaka upitnika (??) Time nas obavještava da nismo unijeli dovoljno brojeva Možemo unijeti i drugi broj i program će se normalno dalja izvršavati

Unesemo li kada računalo očekuje dva broje više brojeva (na primjer 3 4 5 6) računalo će ispisati

```
?EXTRA IGNORED
```

Time nas «ono» obavještava da smo unijeli pravilna brojeva Varijablema A i B pridruženi su prve dva broja koja smo unijeli a ostale brojeve ja računelo ignoriralo

Greška

```
?EXTRA IGNORED
```

ne prekida izvršenje programa za razliku od grešaka s kojima smo se do sada sretali.

U naredbi INPUT možemo navesti i string-varijablu. Tada će računalo kada naiđe na naredbu INPUT očekivati da mu unesemo neki tekst, odnosno računalo će sve što otkucamo shvatiti kao tekst i pridružiti string-varijabli koja je navedena u naredbi INPUT. Pogledajmo to na primjeru (prvo obrišite prethodni program sa NEW)

```
10 INPUT A$
20 PRINT A$
30 GOTO 10
```

U ovom programu nije moguće dobiti obavijest ?REDO FROM START zbog toga što bilo koji tekst koji unesemo može biti pridružen string varijabli. Treba, naravno imati na umu da ukupne dužine stringa ne smije biti veća od 255 znakova.

Kao i za numeričkim varijablama unutar jedne INPUT naredbe možemo navesti više string varijabli odvojenih zarezom. Kada računalo napiše upitnik za svaku od tih string varijabli napisat ćemo po jedan tekst. Tekstove ćemo odvajati zarezom. Ako želimo da u tekstu koji unosimo u naredbi INPUT upišemo i zarez, tada tekst moramo navesti u navodnicima. Navodnike će »oraz« ignorirati. Unutar iste naredbe INPUT možemo navesti i numeričke i string-varijable. Kada odgovaramo na komandu INPUT moramo tekstove i brojeve unositi onim redom kojim su zapisane varijable unutar naredbe INPUT.

Upišite ovaj program

```
10 INPUT A$
20 INPUT A,B
30 B=B-A
40 PRINT A$;" IMA ";S;" GODINA"
50 GOTO 10
```

Pokrenite ovaj program i kad ugledate prvi upitnik upišite svoje ime. Pritisnite (CR) i na drugi upitnik napišite prvo godinu svoga rođenja, a zatim i tekuću godinu (ovu u kojoj amo sada). Vidite da računalo zna ispisati koliko imate godina. To njegovo »znanje« rezultat je programa koji smo mu napisali.

Primijetili ste vjerojatno da je dosta teško zapamtiti što pojedina naredba INPUT očekuje da joj unesemo. Možemo unutar INPUT-naredbe napisat u navodni cima tekst koji će nam pomoći da pravilno odgovorimo

na INPUT naredbe. Naš prethodni program modificirali smo ovako

```
 10 INPUT "KAKO SE ZOVEŠ ";A$
 20 INPUT "KOJE GODINE SI ROĐEN
";A
 30 INPUT "KOJA JE SADA GODINA "
;B
 40 S=B-A
 50 PRINT A$;" IMA ";S;" GODINA"
```

Vid mo da će tekst naveden u naredbi INPUT biti ispisan prije nego što se ispiše upitnik. Ako tekst naveden u naredbi INPUT napišemo u obliku pitanja i ne ze vršimo ga upitnikom, tada će upitnik koji ispisuje na redba INPUT iza pitanja završiti rečenicu.

Naredbu INPUT ne možemo upotrebljavati umjesto naredbe PRINT jer ako ne navedemo niti jednu varija blu nakon teksta koji smo napisali u naredbi INPUT računalo će prijaviti grešku. Isto teko moramo paziti da nam tekst koji želimo štampati dolazi odmah iza naredbe INPUT a ne nakon što smo naveli jednu ili više varijabli. Za odjeljivanje teksta od varijabli u na redbi INPUT umjesto zareze možemo upotrijebiti točka zarez. Međutim, var jable moramo razdvajati zarezom, znak točke zarez između dvije venjable u naredb. INPUT nije dozvoljen.

NAPOMENE U PROGRAMU (REM)

Kada pišemo neki program, pogotovu ako je u pita nju duži program, možemo vrlo lako zaboraviti što smo učinili u pojedinom dijelu programa i zašto smo neku naredbu napisali baš tako kako je napisana. Može nam se desiti da nakon nekog vremena moramo prepravljati program jer se pojavile neke nova potreba koju taj program mora zadovoljiti. Ako je prošlo više tjedana (pa i više mjeseci) otkeko smo napisali program, vrlo teško ćemo razumjeti program koji smo sami napisali. Još je gore ako moramo prepravljati program koji je netko drugi napisao.

Da bi se izbjegli (ili barem ublažili) svi ovi problemi, poželjno je da u sam program ugradimo određene na pomene koje će nam kasnije pomoći da se snađemo u programu. Ova napomene su vrlo važne pa su autori BASICA ugradili posebnu naredbu koja nam služi za čuvanje teksta u programu. To je naredba REM. Napi šite ovaj tekst

```
10 REM *** OVO JE REM NAREDBA ***
```

Pokrenite program s RUN. Vidite da se nije ništa dogodilo. Naredba REM služi nam isključivo za stavljanje teksta u program i nju računalo potpuno ignorira za vrijeme izvršenja programa. Prema tome, izvršenje nekog programa neće se mijenjati ako u programu postoje naredbe REM.

Iza naredbe REM može se nalaziti bilo kakav tekst s dopuštene je i upotrebe svih znakova jer računalo ovaj tekst nikada ne »čita«. Unutar naredbe REM nije moguće učiniti grešku.

S naredbama REM ne treba biti štedljiv, ali ne treba ni pretjerivati. Prevelika količina naredbi REM otežit će čitljivost programa jednako kao da ih i nema.

Tekst koji je upisan u naredbu REM ne treba biti previše dugačak. Ne primjer

```
10 REM *** DIO PROGRAMA ZA IZRA
ČUNAVANJE POVRŠINE KVADRATA ***
```

To je previše dugačka naredba REM i zbog toga nečitka. Ne valjaju ni previše kratke naredbe REM, na primjer

```
10 REM IPK
```

Teško će ća iz ovoga nešto shvatiti da se izračunava površina kvadrata. Za naredbu REM je najbolje da je duga jednu liniju i da sadržava najbitnije podatke o dijelu programa koji opisuje. U danom primjeru najbolje bi bilo napisati

```
10 REM POVRŠINA KVADRATA
```

Još jednom želim naglasiti vrijednost naredbi REM zbog toga što ih mnogi programeri ne stavljaju. To dovodi do toga da imamo programe koje je teško čitati i još teže preprav1jati. U narednom primjeru u naredbi REM opisani su neki dijelovi programa

```
10 REM POVRŠINA KVADRATA
20 INPUT "KOLIKA JE STRANICA ";
S
30 REM RAČUNANJE POVRŠINE
40 P=S*S
50 PRINT "POVRŠINA JE ";P
60 REM SKOK NA POČETAK
70 GOTO 10
```

DONOŠENJE ODLUKA (IF)

Pri programiranju vrlo često nailazimo na probleme o kojima računalo treba donijeti neku odluku. Ne primjer, ako računamo opseg kruga nije moguće da polumjer kruga ima negativnu vrijednost (ne postoje negativne dužine). To znači da u programu moramo testirati da li je polumjer negativan i ako jeste prijaviti da je nastala greška. Za testiranje pojedinih brojeva služit ćemo se dodatnim operatorima. Usporedimo li dva broja moguće su tri različite stanja

- prvi broj je veći od drugoga
- prvi broj je manji od drugoga
- oba broja su jednaka

Za sveko od ova tri stanja imamo odgovarajuće operatore. To su znači veće (>) manje (<) i jednako (=)

Da biste lakše razlikovali znakove manje i veće, zapamtite da je vrh uvijek okrenut prema manjem broju (linije se šire prema većem broju).

Osim ove tri osnovne usporedbe ponekad će nas zanimati i neka od složenih usporedbi. Postoje tri složene usporedbe i to su

- prvi broj je veći ili jednak drugome
- prvi broj je manji ili jednak drugome
- prvi broj je različit od drugoga

Za složene usporedbe imamo složene operatore i to su veće-jednako (>=) manje jednako (<=) različito (<>)

Vratimo se našem problemu negativnog polumjera. Da bismo ustanovili da li je neki broj negativan, poslužit ćemo se spoznajom da su negativni brojevi manji od 0. Pogledajmo

```
10 INPUT "POLUMJER ";R
20 IF R<0 GOTO 10
30 PI=3.141
40 O=2*R*PI
50 PRINT "OPSEG JE ";O
60 GOTO 10
```

Najprije pokrenite ovaj program i unesite neku pozitivnu vrijednost za polumjer R. Vidjet ćete da program ispravno radi. Sada unesite negativnu vrijednost. Kao što vidite, program vas ponovno pita za vrijednost po-

tumjera odnosno na prihvaća negativne vrijednosti. Pogledajmo kako smo to postigli. Nakon što se u naredbi 10 unese vrijednost varijable R, računalo dolazi na liniju 20. U toj liniji nalazi se naredba IF (IF na engleskom znači ako). Iza riječi IF dolazi uvjet. Uvjet je izraz koji sadrži jedan od operatora za usporedbu. Ovdje to je test da li je R manji od 0 odnosno da li je R negativan broj. Iza uvjeta nalazi se naredba koja se izvršava SAMO AKO JE UVJET ISPUNJEN. Dakle, linija 20 možemo pročitati ovako: ako je R manje od 0 onda idi na liniju 10.

Vidjeli smo da računalo pravilno izračunava vrijednost ako je R pozitivan broj. Naredba IF ne izvršava tekst iza uvjeta ako uvjet nije zadovoljen. Jednostavnije rečeno, ako uvjet u naredbi IF nije zadovoljen, izvršavanje programa nastavit će se na prvoj naredbi ispod naredbe IF.

Kao što smo vidjeli, naredbe IF može izazvati uvjetovani skok u programu. Takav uvjetovani skok vrlo često je pr mjenjivan u svim ozbiljnijim programima. U sljedećem primjeru iskoristit ćemo naredbu IF da odaberemo dio programa koji će se izvršiti. Prvo ćemo unijeti dva broja i zatim znak plus ili minus. Ako unesemo plus, prva dva broja će se zbrojiti, ako unesemo minus oduzet će se drugi broj od prvoga. »Ostalo« neće prihvatiti ništa drugo osim plusa i minusa.

```
10 INPUT "PRVI BROJ ";A
20 INPUT "DRUGI BROJ ";B
30 INPUT " + ILI - ";A$
40 IF A$="+" GOTO 100
50 IF A$="-" GOTO 200
60 GOTO 30
100 REM ZBRAJANJE
110 S=A+B
120 PRINT "ZBROJ JE ";S
130 GOTO 10
200 REM ODUZIMANJE
210 S=A-B
220 PRINT "RAZLIKA JE ";S
230 GOTO 10
```

U liniji 40 vidimo da je u uvjetu unutar naredbe IF moguće usporediti i stringove. O tome više u poglavlju koje govori o tome kako »otac« pamti slova.

Vidimo da u linijama 40 i 50 testiramo da li je u string-varijabli A$ plus ili minus. Pazite: ako nije zadovoljen uvjet ni u naredbi 40 računalo odlazi u naredbu 50, a ako nije zadovoljen uvjet ni u naredbi 50 izvršava se naredba 60 odnosno skače se na liniju 30 i ponovno postavlja pitanje o matematičkoj operaciji. Vidimo da će linija 60 biti izvršena jedino ako nisu zadovoljeni uvjeti niti u naredbi 40, niti u naredbi 50. Takve točke u kojima se tok programa mijenja pod nekim uvjetom nazivamo još i granama programa.

Iza uvjeta u naredbi IF ne mora biti naredba GOTO

Ako želimo staviti neku drugu naredbu tada moramo iza uvjeta napisati riječ THEN iza te riječi možemo napisati bilo koju drugu BASIC naredbu i ona će se izvršiti jedino ako je uvjet ispunjen. Iza riječi THEN može stajati i druge naredbe IF međutim time program postaje znatno zamršeniji i teži za praćenje pa se to vrlo rijetko primjenjuje. Prepravit ćemo neš prethodni primjer uz upotrebu riječi THEN u naredbi IF

```
 10 INPUT "PRVI BROJ ";A
 20 INPUT "DRUGI BROJ ";B
 30 INPUT " + ILI - ";A$
 40 IF A$="+" THEN PRINT "ZBROJ
JE ";A+B
 50 IF A$="-" THEN PRINT "RAZLIK
A JE ";A-B
 60 GOTO 30
```

Vidimo da je ovako napisan program kraći od prethodnog primjera. Prema tome oblik IF ... THEN upotrijebit ćemo kad god je to moguće jer time program dobiva na čitkosti. To je zbog toga što nije potrebno eliminirati tok programa kroz nepotrebne skokove. Odmah vidimo što će se izvršiti ako je uvjet ispunjen iza riječi THEN može se nalaziti i naredba GOTO međutim tada THEN nije potrebno pa se možemo izostaviti. U slijedećem primjeru imamo program koji ispisuje brojeve od 1 do 10 ali ne ispisuje broj 7

```
10 FOR A=1 TO 10
20 IF A=7 THEN A=A+1
30 PRINT A
40 NEXT A
```

Vidimo da naredba IF koje se nalazi unutar petlje mijenja vrijednost kontrolne varijable A onda kada A ima vrijednost 7. Zbog toga se vrijednost 7 ne ispisuje.

Osim testiranja određenih stanja pomoću naredbe IF možemo formirati petlju nekih petlji FOR. Pogledajmo primjer

```
10 A=0
20 A=A+1
30 IF A>10 GOTO 60
40 PRINT A
50 GOTO 20
60 PRINT "KRAJ"
```

Vidimo da se u naredbi 30 testira da li je A veće od 10 i ako nije, skače na liniju 20. Tako formirana petlja u potpunosti je nalik na petlju FOR. To je zbog toga što se uvjet testira nakon povećavanje varijable A jednako kao što to čini naredba NEXT u petlji FOR. Osnovne eliđnost između ovakve petlje i petlje FOR je u tome što će nakon završetka petlje broj biti veći od postavljene gornje granice. Pogledajmo primjer

```
10 A=0
20 IF A=10 GOTO 60
30 A=A+1
40 PRINT A
50 GOTO 20
60 PRINT "KRAJ"
```

Ovdje se ispisuju potpuno isti brojevi kao i u prethodnom primjeru jedino što je varijabla A testirana prije nego što je povećana pa će se po izlasku iz petlje imati vrijednost jednaku gornjoj granici određenoj u petlji. Ovaj drugi postupak primjenjujemo mnogo češće pogotovu kada radimo pretraga nekih podataka. Umjesto onog prvog postupka uvijek je bolja raditi petlju FOR.

Pri formiranju petlji upotrebom naredbe IF treba paziti na to da ako se uvjet testira na ulazu u petlju onda je moguće da se petlja ne izvrši niti jednom (ako je uvjet zadovoljan već pri prvom testiranju). Ako uvjet testiramo na kraju petlje onda će se ta petlja izvršiti najmanje jednom bez obzira na to da li je pri prvom testiranju uvjet zadovoljen ili nije. To je ilustrirano u naredna dva primjera

```
10 REM TEST NA ULAZU
20 A=0
30 IF A=0 GOTO 60
40 PRINT A
50 GOTO 20
60 PRINT "KRAJ"
```

```
10 REM TEST NA KRAJU
20 A=0
30 PRINT A
40 IF A=0 GOTO 60
50 GOTO 20
60 PRINT "KRAJ"
```

U prvom primjeru linije 40 i 50 uopće se ne izvršavaju a u drugom primjeru=izvršava se linija 30 bez obzira na to što je već u prvom prolazu zadovoljan uvjet. Dakle sve linije prije testa s kojim izlazimo iz petlje izvršavaju se jeden puta više nego linije iza testa

NAREDBE ZA PREKIDANJE PROGRAMA

Povremeno se javi potreba da izvođenje nekog programa prekinemo kada je u programu nastupilo određeno stanje. To postižemo naredbom STOP čija značenje nije potrebno objašnjavati. Ova naredba može se naći na kraju nekog dijela programa koji se izvršava samo onda kada je počinjena greška ili unutar IF naredbe da bi se prekinulo izvođenje programa na zahtjev korisnika. Naredni program ilustrira upotrebu ove naredbe

```
10 INPUT "UNESI DVA BROJA ";A,B
20 PRINT "NJIHOV ZBROJ JE ";A+B
30 IF A=0 THEN STOP
40 GOTO 10
```

Ovaj program zbraja dva unesena broja i radi to sve dok prvi uneseni broj nije 0. Kada pronađe da je prvi uneseni broj 0 program se prekida uz pomoć naredbe STOP »Oraо« će ispisati ovaj tekst

```
BREAK
15 IF A=0 THEN STOP?
```

Time nas obavještava da je prekid nastupio zbog naredbe STOP a ispisuje nam i liniju u kojoj je nešto tu naredbu. Nakon što je program prekinut naredbom STOP ili smo ga prekinuli pritiskom na (CTL) C možemo nastaviti izvođenje programa ako otkrijemo

CONT

CONT je kratica od continue što ne engleskom znači nastaviti

Naredba STOP ima specifičnu primjenu prilikom testiranja programa. Ako u programu imamo više grananja i pojavljuje nam se jedna greška u nekom od tih grananja a ne znamo u kojem upotrijebit ćemo nekoliko naredbi STOP. U svaku granu programe ubacit ćemo STOP naredbu i zatim pokrenuti program. Kada program nađe na naredbu STOP »oraо« će ispisat liniju odnosno na koju je STOP-naredbu naišao. Sada ćemo pogledati da li se sve zbiva prema planu i ako je sve dobro program ćemo nastaviti sa CONT. Ovaj postupak ponavljat ćemo sve dok ne otkrijemo grananje u kojem je greška

BASIC mikroračunala »oraо« posjeduje i naredbu END. Ova naredba zaostala je iz vremena velikih računala koja su imala mnogo terminala pa se u svakom BASIC-programu moreo označiti kraj programa. Ovu naredbu možemo staviti na kraj našeg programa to neće izmijeniti njegovo izvršavanje. Naredbu END mo-

žemo i pamatnije upotrijebiti. Ako želimo program završiti usred neke petlje bez skakanja na njegov kraj možemo ne tom mjestu napisati END. Kada »orač« naiđe na naredbu END on će postupiti jednako kao i kad izvrši posljednju naredbu u programu.

Program koji smo završili naredbom END na može žemo nastaviti naredbom CONT

Funkcija INT služi nam za dobivanje cijelog broja zaokruživanjem na PRVI MANJI BROJ. To znači da će nam funkcije INT u pozitivnim brojevima davati za rezultat cjelobrojni dio broja odnosno ono što se nalazi ispred decimalne točke. Na primjer ako zatražimo

```
PRINT INT (3.5)
```

računalo će ispisati broj 3. Vidimo da je broj 3 prvi manji cijeli broj ispred broja 3.5

S negativnim brojevima sve je nešto kompliciranije. U primjeru

```
PRINT INT (-3.5)
```

dobit ćemo rezultat -4. Ako se sjetimo skupa cijelih brojeva vidjet ćemo da je broj -4 prvi manji cijeli broj ispod broja -3.5. To treba imati na umu kada se radi s funkcijom INT jer možemo pogriješiti u programu.

Ako želimo pravilno zaokruživanje broja uz funkciju INT moramo prvo na broj pribrojiti vrijednost 0.5. To slijedi z pravile o zaokruživanju brojeva. Svi brojevi čiji decimalni dio ma vrijednost manju od 0.5 zaokružuju se nadolje a svi oni kojima je decimalni dio veći ili jednak 0.5 zaokružuju se nagore. Kada na neki broj pribrojimo 0.5 a njegov decimalni dio je bio manji od 0.5 cjelobrojni dio broja neće se promijeniti. Ako 0.5 pribrojimo na broj čija je decimalni d o već ii jednak 5 cjelobrojni dio broje povećat će se za jedan. U oba slučaja funkcija INT dat će nam nakon pribrajanja 0.5

FUNKCIJA INT

Ovo je prva iz skupine funkcije o kojima će kasnije biti mpogo više govora. Za početak recimo samo da je svim funkcijama zajedničko to što se iza njih navodi jedan podatak u zagradi (to se nazive argument funkcije) i one daju određeni rezultat koji možemo ispisati ili pridružiti varijabli. Zbog toga funkcije ne možemo navoditi samostalno u trenutnu BASIC-programe već mo raju biti unutar nekog izraze

željeni rezultat. Te možete vidjeti u narednom programu u koji možete unositi bilo koji decimalni broj i program će ga pravilno zaokružiti

```
10 INPUT A
20 PRINT INT (A+.5)
30 GOTO 10
```

U ovom primjeru vidimo također da je broj 0.5 u liniji 20 naplaćan kao 5 što mikroračunalo dopušta. Možda ste primjetili da »oreo« sve brojeve koji su manji od 1 ispisuje bez nule ispred decimalne točke.

Osim zaokruživanja na cijele brojeve ponekad nam je potrebno zaokruživanje na određeni broj decimalnih mjesta. U primjeru zaokruživanja na dvije decimale objasnit ćemo kako se to radi. Ako želite zaokruživanje na tri decimale upotrijebit ćete umjesto broja 100 broj 1000, za četiri decimale broj 10 000 itd

```
10 INPUT A
20 A=A*100
30 A=A+.5
40 A=INT(A)
50 A=A/100
60 PRINT A
70 GOTO 10
```

Pravilno ovaj primjer detaljno. Nakon što smo u liniji 10 unijeli neki broj (na primjer 9.6666) u liniji 20 pomnožili smo taj broj sa 100 (666.66). Na ovako dobiveni broj primjenili smo već prije opisani postupak zaokruživanja. Dakle prvo smo pribrojili 5 (667.16) a zatim potrežili INT od tog broja (667). Na kraju smo u liniji 50 podijelili taj broj sa 100 (6.67). Vidimo da smo na kraju dobili broj ispravno zaokružen na dvije decimale.

Funkciju INT možemo korisno upotrijebiti i onda kada želimo ispitati da li je neki broj cijeli broj. U narednom primjeru ispred cijelih brojeva štampane su zvjezdice. To je postignuto u naredbi 20 gdje se kontrolira da li je broj A jednak INT(A). Ako jeste, u pitanju je cijeli broj.

```
10 FOR A=0 TO 10 STEP .5
20 IF A=INT(A) THEN PRINT "*";
30 PRINT A
40 NEXT A
```

Osim za provjeru da li je u pitanju cijeli broj funkciju INT možemo upotrijebiti i za to da ispitamo da li je neki broj djeljiv s nekim drugim brojem. Sve se izvodi slično odnosno prvo ćemo podijeliti broj koji ispitujemo a zatim ćemo provjeriti da li smo dobili cjelobrojni rezultat. U narednom primjeru program ispisuje brojeve od 1 do 50 ali ne ispisuje brojeve koji su djeljivi sa četiri

```
10 FOR A=1 TO 50
20 B=A/4
30 IF INT(B)=B THEN GOTO 50
40 PRINT A
50 NEXT A
```

SLUČAJNI BROJ (RND)

Ovo je također riječ koja pripada u kategoriju funkcija. Pozabavit ćemo se tom funkcijom koje omogućuje mikroračunalu da «zamisli» neki slučajni broj. Da bi smo pošteni moramo reći da računalo ne može izmisliti broj. Zapravo «oko» izračunava jedan broj uzimajući pri tome početnu vrijednost koju nam nije poznata i primjenjujući toliko složen obrazac da se nama čini kao da se brojevi slučajno pojavljuju. Ove funkcije odnosno slučajni brojevi nužni su za istraživanje dinamičkih procesa, testiranje matematičkih i statističkih programa i slično. Naravno slučajni brojevi imaju ši roku primjenu u igrama. Unesite ovaj program

```
10 FOR A=1 TO 10
20 PRINT RND (7)
30 NEXT A
```

Vidimo na ekranu ispis deset različitih brojeva. Pogledajmo prvo liniju 20 koja je ispisivale te brojeve. U njoj nalazimo funkciju RND i iza nje broj sedam u zagradi. Ovaj broj potrebno je napisati iako njegove vrijednosti nije bitna. To može biti bilo koji pozitivan broj različit od nule. Ovaj broj «oru» služi prilikom izračunavanja slučajnog broja, ali to ne znači da će isti broj dati uvijek isti rezultat. Jedino nula davat će stalno isti rezultat odnosno ponavljat će posljednji generirani slučajni broj.

Pogledajmo sada brojeve koje nam je program ispisao. Vidimo da su svi brojevi manji od jedan. Funkcija RND generira nam slučajni broj u rasponu od nule do jedan koji može biti nula ali ne može biti jedan. Matematičari bi to napisali ovako

$$0 <= RND < 1$$

Uz pomoć jednostavne matematičke operacije i funkcije INT možemo dobiti broj u nekom drugom raspo-

nu. U sljedećem primjeru generirat ćemo brojeve u rasponu od 0 do 5

```
10 FOR A=1 TO 10
20 B=5*RND (7)
30 PRINT INT(B)
40 NEXT A
```

Vidimo da nam ovaj program ispisuje brojeve u rasponu od 0 do 5. To smo postigli tako da smo slučajni broj u rasponu od 0 do 1 pomnožili sa brojem koji je ZA JEDAN VEĆI OD GORNJE GRANICE INTERVALA. Taj broj je za jedan veći zbog toga što je funkcija INT kao što smo rekli zaokruživanje na prvi manji cijeli broj. Prema tome kada god množimo slučajni broj s nekim drugim brojem moramo imati u vidu učinak funkcije INT.

Ako nam treba broj u nekom drugom rasponu (ne od nula) tada ćemo slučajni broj pomnožiti s razlikom između donje i gornje granice intervala uvećanom za jedan. Nakon toga pribrojit ćemo donju granicu i brišemo uzeti INT od dobivenog rezultata. Ako je donja granica A a gornja granica B to možemo napisati ovako

```
INT ((B-A+1)*RND(7)+A)
```

Pogledajte to na primjeru: treba nam slučajni broj u rasponu od 3 do 9. Oduzet ćemo broj 3 od broja 9. Ne dobiveni rezultat (6) pribrojit ćemo 1 i dobiti broj 7. Ovaj broj pomnožit ćemo sa RND 1 na kraju pribrojiti donju granicu (3). Tako dobiveni rezultat pretvorit ćemo u cijeli broj pomoću funkcije INT. To je ilustrirano primjerom

```
10 FOR A=1 TO 10
20 S=INT((9-3+1)*RND(1)+3)
30 PRINT S
40 NEXT A
```

U liniji 20 možemo jednostavnije napisati INT (7*RND (1)+3)

Najčešće će nam biti potrebni brojevi u intervalu od 1 do nekog broja. Tada se sve pojednostavljuje jer nam je dovoljno da prije nego što potražimo cjelobrojni dio RND pomnožimo s gornjom granicom i dodamo 1. Slijedeći primjer ispisuje brojeve u rasponu od 1 do 100.

```
10 FOR A=1 TO 20
20 S=INT(100*RND(1)+1)
30 PRINT S
40 NEXT S
```

OPIS I PRAVILA IGRE

Cilj je igre da pogodimo broj koji je »zamislilo« računalo. Taj broj nalazi se u rasponu od 1 do 100 e može biti i 1 i 100. Ne svaki naš pokušaj de pogodimo broj računalo će ispisati da l je broj koj treba pogodni veći ili manji od broja koji smo sami unijeli a ako smo pogodili ispisat će nam broj pokušaja koji nam je bio potreban da dođemo do točnog rješenja.

Nakon što smo proučili pravila igre, možemo pisati program. Zadatak ćemo razbiti na manje cjeline i nakon toga svaku cjelinu obraditi posebno.

PISANJE PROGRAMA

Ako ste pažljivo pratili sve što je do sada objašnjeno steki ste dovoljno znanja da možete samostalno pisati programe. Dosadašnji dio knjige bavio se u prvom redu pojedinim BASIC-instrukcijama. Sada treba progovoriti nešto o tome kako se programi pišu, kako se za taj posao pripreme i kako se programi dokumentiraju.

Da bismo to ilustrirali proći ćemo zajedno čitav posao pisanja nekog programa. Uzet ćemo si u zadatak da napišemo kratku BASIC igru. Posao započinjemo tako da razmotrimo sve što znamo o samom zadatku. U igri to je opis igre i pravila igranja.

DIJELOVI PROGRAMA

A. generiranje slučajnog broja u rasponu od 1 do 100 i postavljanje brojila pokušaja na 0

B – unošenje broja i usporedivanje sa zadanim brojem

C – obavještavanje o rezultatu pokušaja

D – ispisivanje broja pokušaja

E – završni dijelovi igre

Ove dijelove razradit ćemo u još manje skupine. Svaku od ovako nastalih cjelina poslije ćemo prevesti u BASIC naredbe. Takav prikaz programa nazivamo skicom programa.

SKICA PROGRAMA

A 1 – generiranje slučajnog broja

A.2 – postavljanje brojila pokušaja na 0

A 3 – ispisivanje uvodne poruke

B 1 – unošenje broja

B 2 – povećanje brojila pokušaja

B.3 – ispitivanje da li je broj jednak zadanom, ako je idi na D 1

B 4 – ispitivanje da li je broj manji od zadanog ako je idi na C 1

B.5 – idi ne C 3 (ovdje smo sigurni da je broj veći od zadanog jer nije ni manji ni jednak)

C 1 – ispisivanje poruke da je broj manji od zadanog broja

C.2 – idi na B 1

C 3 – ispisivanje poruke da je broj veći od zadanog broja

C.4 – idi na B.1

D 1 – ispisivanje poruke da je broj uspješno pogođen

D 2 – ispisivanje broja potrebnih pokušaja

E 1 – ispisivanje poruke da li se želi igrati još jednom

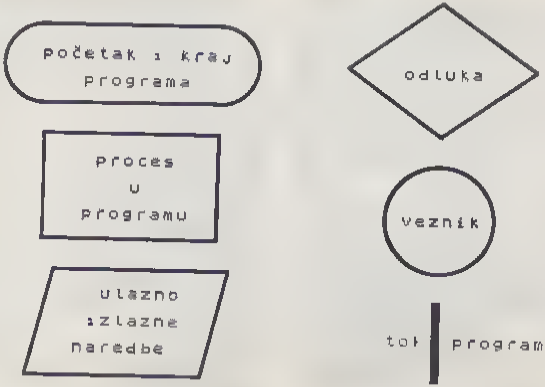
E 2 – unos odgovora

E.3 – ispitivanje da li je odgovor potvrdan ako je idi na A 1

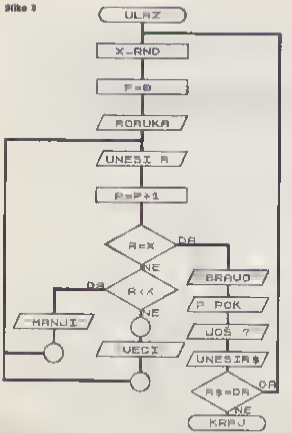
E 4 – kraj programa

Skicu programa možemo prikazati grafički. To je sva kako najbolji put da se prikaže i točno izradi skica programa. Tako izrađenu skicu programa nazivamo dijagram toka programa ili blok-dijagram. Pri izradi ovih dijagrama služimo se određenim grafičkim simbolima koji su prikazani na slici 2. Na slic 3 vidimo dijagram toka naše igre. Vidimo da svakom elementu skice programa odgovara jedan grafički element na dijagramu toka. Toku programa u tom dijagramu odgovaraju linije koje povezuju pojedine blokove. Na mjestima gdje donosimo određene odluke linije se granaju i zato ove točke nazivamo granjanjem programa.

Slika 8


početak i kraj programa
proces u programu
ulazno izlazne naredbe
odluka
veznik
tok programa

Slika 3



Kada smo završili izradu slike programa i dijagrama toka, možemo pristupiti prevođenju (neki ovo nazivaju kodiranje) programa. Svaki element slike programa prevest ćemo u jednu ili više naredbi. U ovom primjeru slika je izrađena tako detaljno da svaku blok možemo prevesti u samo jednu BASIC-liniju. Evo kako izgleda listing programa:

```
 10 X=INT (100*RND(7)+1)
 20 P=0
 30 PRINT "POGODI BROJ OD 1 DO 1
00"
 40 INPUT A
 50 P=P+1
 60 IF A=X GOTO 130
 70 IF A<X GOTO 90
 80 GOTO 110
 90 PRINT "TVOJ BROJ JE MANJI OD
MOGA"
 100 GOTO 40
 110 PRINT "TVOJ BROJ JE VEĆI OD
MOGA"
 120 GOTO 40
 130 PRINT "BRAVO! USPIO SI!"
 140 PRINT "TREBALO TI JE";P;"PO
KUŠAJA"
 150 PRINT "ŽELIŠ LI IGRATI JOŠ
JEDNOM";
```

```
160 INPUT A$
170 IF A$="DA" GOTO 10
180 PRINT "DOVIDENJA"
```

Ako budete ovaj program unosili u računalo prepisivat ćete ga iz knjige. Zbog toga ne postoji potreba da program prvo napišete na papiru. Kada budete pisali svoj program najbolje je da skicu programa prevedete u BASIC naredbe najprije na papiru pa da tek onda unesete program u memoriju. To napominjem zbog toga što ćete tako izbjeći mnoge greške koje bi inače nastale pri direktnom upisivanju programa. Osim toga n naredbama 60 70 i 80 izvodi se skok na naredbe koje se nalaze ispod ovih naredbi. Ako program pišete direktno u računalo vi dok pišete liniju 60 ne znate de u njoj treba napisati skok na liniju 130 jer je još niste ni napisali. U radu na papiru vrijednosti n GOTO naredbama upišu se tak nakon što su napisane sve linije programa.

U ovaj program nisam uvrstio niti jednu REM-naredbu. Učinio sam to zbog toga da prevođenje skrce programa u tekst programe bude doslovno. Između linije 20 i 30 mogli smo dodati n naredbi REM tekst POČETAK PETLJE. Između linija 120 i 130 može se napisati REM USPJEŠAN POGODAK i tako dalje.

Sve što ete izradit prilkom izrade programe (opis skica dnagram toka) zajedno s papirima na koj me ste prevodili program u BASIC-naredbe jest dokumentacija programa. Kruna dokumentacije je listing programe ako možete da vaše računalo poveznete na štampač. Ne takav listing treba dodat određene komentara. To treba učiniti naposredno nakon završetka rada na programu dok se još sjećamo svega što emo na programn radili. Tako kompletiranu doknmentaciju treba sačuvati jer će nam ona pomoći eko se javi bilo kakva potreba da n programn nešto promijenimo. Kada su u pitanju igre male je vjerojatnost da ćemo u njima nešto mijenjati nakon što su jednom završene. ali se može desiti da ih prenosimo na neko drugo računalo r tede će nam ponovno biti potrebna dokumentacija. Najčešće je mnogo lakše prstupiti izradi novog programe nago mi dificiirati program za koji na postoji prikladne dokumentacije.

Razlike između dobrog i odličnog programera najčešće je u tome što odličan programer ime dobru dokumentaciju za svoje programe.

Mnogi će vam savjetovati da je najbolje za izradu programa da sjednete za računalo i sve radite d rektno iz glave. Činjenica je da to može 1% programere al je isto tako činjenica da 99% programere mlali da to može. Nadam se da ćete vi biti bilo u onih 1% koji to zaista mogu. Ili barem u onih 1% koji cijelom poslu pristupaju pametno.

Kada se program piše direktno n računalo tada se sve otkrivene greške popravljaju stihijski i bez mnošenja n bilo kakvu dokumentaciju. To rezultira programom koji je potpuno nepregledan koji n sebi sadržava tko

zna koliko grešaka koje u toku testiranja nisu otkrivene odnosno to je najčešće vrlo loš program Takve programa zbog njihove zapetljanosti programeri nazivaju »špageti-programi«

Posebna stavka u radu na programu jest ispitivanje programa To se radi onda kada je program kompletno gotov U ispitivanju programa ima dva pristupa Prvi je pristup da program pustimo u rad i tast ramo ga u radu To je tzv test u praksi I on će najvjerojatnije otkriti bilo kakvu grešku ako ona postoji u programu Međutim takvo testiranje zahtijeva vrlo mnogo vremena a ponekad i mnogo novaca ili je čak opasno Zamislite da jedan program koji kontrolira kretanje podzemnih željezn ca u nekom velegradu testiraju tako da ga puste da radi pa šta bude — bude

Drugo testiranje je da program ispitujemo blok po blok unoseći u svaki blok sve moguće ulazne podatke koji u taj blok mogu doći u toku rada programa Kad ispitamo svaki pojedini element, ispitivat ćemo kako ti elementi rade u međusobnom odnosu a tek onda na kraju i cijeli program

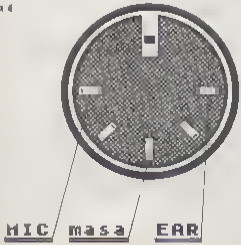
Testiranje programa je vrlo važan dio rada ne programu jer program postaje smislena cjelina tek nakon što iz njega uklonimo greške Na treba se zanositi idejom da možemo pisatr programe bez grešaka jer praksa pokazuja da r programeri s vrlo dugim programerskim skustvom povremeno griješe ponekad čak i najbanalnije Za vrijeme testiranje programa ne greške ne treba gledati kao na osobnu uvredu već svom programu treba pristupiti koliko god je to moguće objektivno i kritički Dobro je ako testiranje programa mo žemo povjeriti nekom drugom jer će osoba koja nije radila program neke greške otkriti znatno lakše od autora programe

Svaku grešku koju otkrijamo odnosno ispravak grešk r općenito svaku promjenu u programu trebe odmah unijeti u programsku dokumentaciju

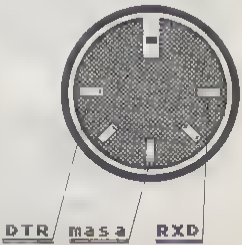
ČUVANJE PROGRAMA

Kada bismo morali svaki puta iznova upisivati program u računalo kada nam je on potreban na bi bilo velike koristi od mikroračunala. Zbog toga su konstruktor računala predvidjeli mogućnost da se programi spremaju na magnetska vrpce. Mi ćemo upotrijebiti

Slika 4



Kasetofon



Štampač

običan kućni kasetofon i standardne muzičke kasete. Ne sliči 4 vidimo kako treba povezati »orao« s kasetofonom. Nakon što smo sve povezali moramo na kasetofonu pravilno podesiti glasnoću i boju tona. Za glasnoću je najbolje da namjestimo potenciometer negdje oko sredine. Potenciometar za boju tone treba postaviti na visoke tonove. Ze snimanja programa najbolje je uzeti nove vrpce. Važno je da na vrpci neme oštećenja jer program neće biti dobro spremljen.

Ako u memoriji imamo program (recimo_ igru iz prethodnog poglavlja) možemo ga spramiti na vrpcu. Povezat ćemo »orao« s utičnicom MIC na kasetofonu (drugi vod nečemo spojiti jer to pravi probleme u mnogim kasetofonima). Neplest ćemo

```
SAVE "IGRA"
```

Tekst u navodnicima je ime programa i mi ga potpuno slobodno pridružujemo programu. Jedino je ograničenje da ime ne smije biti duže od tih znakove. To ime služi nam da kesnije lakše prepoznamo program na vrpci.

Uključimo snimanje na kasetofonu i pritisnemo (CR). Iz »orla« ćemo začuti pištav zvuk što je znak da se program anima ne vrpcu (zapravo program se snima u obliku tog tona). Ako je program koji snimemo duži od 255 bytsove primijetit ćemo da se pojavlio više odvojenih blokova zvuka. To je zbog toga što »orao« program spreme ne vrpcu u obliku više uzastopnih blokova od kojih je svaki dug 255 bytsova. Kad nam se pojavi kursor sa menje programa je završeno. Tako snimljeni program možemo uvijek ponovno unijeti u memoriju. To radimo ovako

To radimo oveko

Povežemo »ora« s utičnicom EAR na kasetofonu (opet je dobro drugi vod isključiti). Premotamo vrpcu na početak programa. Upišemo

```
LOAD "IGRA"
```

Nakon što pritisnemo (CR) «orao« će ispisati

```
SEARCHING "IGRA
```

a zatim će redom ispisivati sve programe na koje nailde na vrpci sve dok ne nailda ne program «IGRA« Iza imena svakog programa ispisat će tip programa i to slovo B za BASIC-programe slovo O za blokove spremljene memorije_a slovo D za podatke. Iza ovih slove doći će broj koji označava broj bloka u pojedinom programu

```
LOAD " '
```

Tada će »orao« uzeti prvi program nie koji nailda na vrpci

Ako prilikom učitavanja nekog bloka nastupi greška, »prao« ce nas o tome obavijestiti i zahtijevati da vrpcu vratimo nazad kako bi ponovno pokušao učitati blok

Ako se program ne može učitati ili ako se učitava ali je oštećen najvjerojetnije imamo problema s nepravilno podešenom glasnoćom i visinom tone. Previsne vrijednosti najlakše ćemo pronaći eksperimentiranjem

Na kasetu treba zapisati koje smo programe na nju snimili - ako kasetofon ima brojilo možemo napisati i broj od kojeg program počinje

POTPROGRAMI

Pri pisanju dužih programa desava se da ne nekoliko mjesta ponavljamo isti sklop naredbi. Na primjer: više puta u istom programu računamo neke parametre. Nije korisno svaki puta iznova pisati isti sklop BASIC-na redbi jer to zauzima mnogo prostora u memoriji (program je bez potrebe duži) i odnosi mnogo vremena. Da bismo to izbjegli služimo se potprogramima (subrutine).

Potprogram je dio programa napisan tako da ga možemo po volji mnogo puta pozvati iz bilo kojeg dijela programa. U sljedećem primjeru vidimo program u kojl unesemo dva broja a zatim znak da li zelimo dijeljenja ili oduzimanje. Program bez obzira kojim redom unesemo brojeve oduz ma manji broj od većega, odnosno dijeli veći broj s manjim

```
10 INPUT "UNESI DVA BROJA ";A,B
20 INPUT " - ILI / ";A$
30 IF A$="-" GOTO 80
40 IF A$<>"/" GOTO 20
50 GOSUB 110
60 PRINT "KVOCJENT JE ";A/B
70 GOTO 10
80 GOSUB 110
90 PRINT "RAZLIKA JE ";A-B
100 GOTO 10
110 REM POTPROGRAM
120 IF A>B GOTO 160
130 T=A
140 A=B
```

```
150 BeT
160 RETURN
```

Program je vrlo nalik na primjer koji smo već imali Novo je u njemu uvođenje naredbi GOSUB u linijama 50 i 80 i naredbe RETURN u liniji 160

Naredba GOSUB ima vrlo sličan efekt kao i naredba GOTO Jedina je razlike što »oreo« kade naide na naredbu GOSUB »zapamti« broj linije u kojoj je nalaza na ovu naredbu pa tek onde izvede skok Tok se zatim nastavlje od linje na kojoj je izveden skok i teče nor malno sve dok i oreo« ne naide na naredbu RETURN Naredbe RETURN znači » vrati se« i naređenje je ra čunalu da se vrati ne mjesto odakle je pozvano s naredbom GOSUB De bi se vratio oreo« uzme broj linije koji je prethodno zapamtio i nastavi na prvoj liniji ispod

Vid mo da smo upotrebom ove naredbe u našem primjeru izbjegli potrebu da dva puta isplsujemo linije 120 do 150 Ako program pišemo tako de ze svaki od sitnih poslova koje treba obaviti napišemo potprogram e zatim napišemo glavni program koji će pozivati te potprograme dobit ćemo strukturirani program. Strukturirani programi mnogo su lakši za pisanja bitno pregledniji i što je najvažnije svaki od potprograma možemo nezavisno testirati To je razlog de potpro greme pišemo ćak i onda kada ih pozivamo samo jednom Ako potprograme pravilno dokument-ramo možemo ih čuvati odvojeno i zatim kada radimo novi program jednostavno u njega uključimo već gotove potprograme Na primjer potprogram ze crtanje elipse možemo jednako dobro iskor stiti u programu za crta nje kao i u programu za učenje geometrije

Sveki program u načelu sadrževe sve obilježja pre grama Zbog toge potprogram možemo raditr jednake ko ste smo redir program Dakle za njega možemo izradit posaban opis loke potrebne dijegrame i tek nakon što ga up šemo testiramo možemo ga uvrsul u glavni program

Posvetimo se malo linijama 130 do 150 Rekli smo da bez obzira ko m redom unesemo brojeve moramo oduzeti manj od većega Zbog toge u liniji 120 testi ramo da l je A veći od B Ako jeste sve je u redu si ako nije moramo sadržaj koji se nalazi u varijabli A staviti u varijablu B a sadržaj iz varijable B staviti u varijablu A Jednostavnije moramo zamijeniti varijable Zamislite da imate dvije boce U jednoj se nalazi voda a u drugoj vino Jasno je da istovremeno ne možemo pretresti vino u onu bocu u kojoj je voda i vodu iz te boce u onu u kojoj je bilo vino Zbog toge ćeme se poslužiti trećom bocom r privremeno u nju pretočili ti vino ili vodu Slično ćemo postupati kada moramo zami jeniti vrijednost dvije varijable Najpre cemo u treću varijablu (u našem primjeru T) smjestiti vrijednost jedne od dvije varijable koje mijenjamo Nakon toga ćeme u tu varijablu smjestiti vrijednost druge varijable a potom sadržaj druge varijable uzeti iz one privremene Za pamtite taj postupak jer se vrlo često primjenjuje u

programiranju e pogotovu u sortiranju
Upišite program

```
10 A=A+1
20 PRINT "POZIV BROJ";A
30 GOSUB 10
```

Ovaj program poziva sam sebe. To znači da »oraz« stalno pamti adresu e koje je pozvan potprogram. To je vrlo ozbiljna greška i zbog toga su konstruktori »orla« ograničili broj poziva na 24. Vidimo da je »oraz« kada smo pokušali pozvati potprogram dvadeset i peti put prijavio grešku. Mi možemo iz jednog potprograma pozvati drugi iz ovog treći i tako dvadeset i četiri puta. Kažemo da se u potprogramima može ući u dubinu od dvadeset i četiri nivoa. To ograničenje neka vas ne zbrinjava jer u praksi vrlo rijetko treba više od četiri nivoa programa. Prema tome dvadeset i četiri nivoa je znatno više nego što će vam ikada zatrebati.

Da ne bi bilo zabune oko tog broja dvadeset i četiri sve možemo promatrati i ovako »Oraz« može »zapamtiti« dvadeset i četiri povratne adresa. Svaki put kada pozovemo neki potprogram »oraz« zapamti jednu adresu. Svaki put kada se vratimo iz potprograma »oraz« »zaboravi« jednu adresu i oslobodi se prostor za pamćenje nove adrese. Prema tome u jednom programu možemo imati koliko god želimo poziva pod uvjetom da ne idemo u više od dvadeset i četiri nivoa.

Kada »oraz« naiđe na naredbu RETURN a da prije toga nije pozvan u potprogram tada je sasvim sigurno u pitanju greška i rad toga će nas on obavijestiti tekstom

```
?DM ERROR
```

i ispisat će nam u kojoj liniji je naišao na RETURN. Takve je greške vrlo često kada napišemo potprogram u nastavku glavnog programa. Tada računalo nakon što je završilo izvođenje glavnog programa nastavi rad u potprogramu i »shvati« to tek kad naiđe na naredbu RETURN. Zato je najbolje glavni program završiti naredbom END pa greške neće biti

Za naredbu GOSUB vrijedi sve ono što smo rekli za naredbu GOTO. Preme toma naredba GOSUB može se naći iz naredbi IF jedino što je tu potrebno ispred naredbe GOSUB napisati riječ THEN. Pogla dajmo to na primjeru

```
10 INPUT A
20 IF A=1 THEN GOSUB 100
30 IF A=2 THEN GOSUB 200
40 IF A=3 THEN GOSUB 300
50 GOTO 10
100 PRINT "JEDAN"
110 RETURN
```

```
200 PRINT "DVA"
210 RETURN
300 PRINT "TRI"
310 RETURN
```

Vidimo da smo u ovom primjeru pozvali tri različita potprograma u ovisnosti o vrijednosti varijable A. Za svaku vrijednost morali smo upisati po jednu naredbu IF. To nije problem kada radimo s tri ćaliri ili čak deset potprograma. Međutim ako treba pozvati dvadesetak i više potprograma, morali bismo upisati veliki broj IF-naredbi. Zbog toga su konstruktori mikroračunala »prisilili« predvidjeli naredbu koja nam omogućuje da ovo testiranje i skok izvedemo u samo jednoj liniji. Pogledajmo kako bi naš prethodni primjer izgledao ako upotrijebimo tu naredbu:

```
10 INPUT A
20 ON A GOSUB 100,200,300
30 GOTO 10
100 PRINT 'JEDAN"
110 RETURN
200 PRINT "DVA"
210 RETURN
300 PRINT "TRI"
310 RETURN
```

U liniji 20 vidimo naredbu ON A GOSUB 100 200 300. Ova naredba uzme vrijednost varijable koja je u njoj navedena i skoči na (pražnji potprogram. U naredbi 20 navedeni su brojevi 100 200 i 300. Ako varijabla A ima vrijednost jedan uzet će se prvi broj iz popisa. Ako je A dva uzet će se drugi broj itd. Ako A ima vrijednost manju od jedan ili veću od broja brojeva u listi izvest će se sljedeća instrukcija iza naredbe ON.

Ako varijable u naredbi ON ima decimalnu vrijednost, računalo će najprije potražiti INT od te varijable. Dakle za svaki broj veći ili jednak jedan a manji od dva »osjetit« će skočiti na prvi broj naveden u listi.

U istoj naredbi može stajati i komanda GOTO umjesto GOSUB. Dakle dozvoljena je formulacija

```
20 ON A GOTO 100,150,200
```

Naredba ON je vrlo moćno »oruđe« za rad jer je moguće uz minimalne izmjene potpuno izmijeniti tok programa. Dovoljno je promijeniti brojeve navedene u naredbi ON.

Najveća vrijednost koju možemo nevesti u varijabli koja odlučuje o skoku u naredbi ON je 255 odnosno u naredbi ON možemo nevesti najviše 255 adresa. Ako je vrijednost varijable veća od 255 ili manja od nule računalo će prijaviti grešku. Zbog toga je ponekad potrebno prije naredbe ON s naredbom IF ispitati ova

vrijednosti i ako je broj izvan predvidenih granica izvesti skok na prvu liniju iispod naredbe ON. Time smo postigli isti učinak kao da naredba ON prihvaća bilo koju vrijednost u varijabli.

S obzirom na to da je maksimalna dužina jedne linije u BASICU 72 znaka može nam se desiti da nam ne stanu sve vrijednosti koje želimo napisati u naredbi ON. Tada ćemo upotrijebiti dvije naredbe ON. U prvoj ćemo navesti onoliko adresa koliko nam stane, zatim ćemo u narednoj liniji varijablu koja odlučuje o skoku umanjiti za onoliko koliko smo adresa naveli u prvoj naredbi ON. Nakon toga stavit ćemo novu naredbu ON u kojoj ćemo normalno nastaviti i stu. To je ilustrirano u narednom primjeru.

```
10 INPUT A
20 IF A<1 GOTO 10
30 IF A>255 GOTO 10
40 ON A GOTO 100,200,300
50 A=A-3
60 ON A GOTO 400,500,600
70 REM NASTAVAK PROGRAMA
```

Ovaj program nije kompletan i napisan je jedino zato da posluži kao primjer. U naredbi 40 mogli se navesti svih šest adresa, ali sam želio da vidite kako se nastavlja lista adresa u naredbama ON.

PODACI UNUTAR PROGRAMA

U nekim programima potrebno je nakon što se program pokrene postaviti nekoliko varijabli koje će nam služiti u toku programa. Možemo varijable definirati naredbom za pridruživanje vrijednosti, ali ako je riječ o većem broju varijabli to će značajno povećati dužinu programa. Ako trebamo na početku programa definirati pet varijabli možemo to učiniti ovako:

```
10 A=1
20 B=5
30 C=7
40 PI=3.141
50 X=255
60 PRINT A,B+C,X*PI
```

Vidimo da smo za definiranje pet varijabli utrošili pet BASIC linija. Isti primjer možemo napisati i ovako

```
10 READ A,B,C,X,PI
20 PRINT A,B+C,X*PI
30 DATA 1,5,7,255,3.141
```

Ovdje smo dobili znatno kraći program. To smo postigli upotrebom naredbe READ. Kada računalo naiđe na ovu naredbu potraži prvu naredbu DATA iz nje pročita brojeve koje pridruži varijablama navedenim u naredbi READ. Prr tome se prvoj varijabli pridružuje prva vrijednost iz naredbe DATA drugoj druge i tako dalje.

Osim što je ovako napisan program kraći mnogo je lakše prom jeniti vrijednosti koje se pridružuju varijablama. Ako u prvom primjeru želimo promijeniti ove vrijednosti moramo mijenjati sva tri linija od 10 do 50. U drugom primjeru dovoljno je promijeniti samo onu liniju u kojoj se nalazi naredba DATA.

DATA naredba je specifična po tome što je sasvim svejedno gdje se nalazi u programu. Kada računalo naiđe na naredbu READ pretražuje program počevši od prve linije sve dok na nađe naredbu DATA. Kada tokom izvršenja programe računalo naiđe na naredbu DATA ono je preskače potpuno jednako kao naredbu REM. Vidimo da naredbu DATA možemo smjestiti bilo gdje u programu i da to nam utječe na izvršavanje programa. Zbog toga ćemo naredbe DATA smjestiti ili neposredno ize naredbi READ ili na sam kraj programa. Češća je prakse da se naredbe DATA smještaju na kraj programa.

Upišite u računalo ovaj primjer

```
10 FOR I=1 TO 5
20 READ A,B
30 PRINT A;"+";B;"=";A+B
40 NEXT I
50 DATA 7,5,8,4,5,3
60 DATA 9,1,2,6
```

Vidimo da u ovom primjeru pet puta izvršavamo naredbu READ i da svaki put učitavamo po dvije varijable. Isto tako vidimo da su varijablama A i B pridruživane vrijednosti onim redom kojim su navedene u naredbama DATA. Kada se Izi program počeo izvršavati računalo je u prvom re lasku na naredbu READ potrebio prvu naredbu DATA i našlo je u liniji 50. Zato su u prvom prolasku varijablama A i B pridružene vrijednosti 7 i 5. Računalo je nakon što je pročitalo ove dvije vrijednosti ispisalo »ZAPAMTILO« koliko je brojeve iz te naredbe DATA pročitano. U sljedećem prolasku na naredbu READ čitanje je nastavljeno iz iste naredbe DATA počevši od prvog broja koji do sada nije pročitan. Kada se nakon trećeg prolaska iscrpile naredbe DATA

u liniji 50 »oraо« je automatski potražio prvu sljedeću naredbu DATA i čitanje nastavio iz nje. Kažemo da računalo posjeduje pokazivač koji pokazuje koji je sljedeći podatak u naredbama DATA na redu za čitanje. Svaki put kada se pročita jedna vrijednost pokazivač se premjesti na sljedeći podatak. Taj pokazivač nazivamo još i data pointer.

S obzirom na to da su podaci u naredbi DATA najčešće gomila nepreglednih brojeva a jako se često dešava da je potrebno promijeniti neki podatak unutar naredbe DATA poželjno je da ove naredbe nisu preduge. Također treba nastojati da se pomoću naredbi REM opišu podaci koji su u naredbama DATA. Ako iz naredbi DATA učitavamo više različitih podataka nije loše u REM naredbama opisati i redoslijed podataka unutar naredbi DATA.

Upišite sada ovaj program

```
10 READ A,B
20 PRINT A,B
30 READ A,B
40 PRINT A,B
50 DATA 1,2
```

Kada pokrenemo program računalo će naiskom na prvu naredbu READ pročitati dvije vrijednosti iz naredbe DATA i pokazivač će pokazivati kraj te naredbe. Kada se nađe na sljedeću naredbu READ računalo će prijaviti grešku jer nema više podataka u naredbi DATA. Dakle, pri pisanju podataka programe moramo paziti da nam se ne desi da čitamo više podataka nego što ih u naredbi DATA ima. Najbolje sada prethodni primjer ovako

```
10 READ A,B
20 PRINT A,B
30 RESTORE
40 READ A,B
50 PRINT A,B
60 DATA 1,2
```

Vidimo da se taj program ispravno izvršava i vidimo da su dva puta čitani isti podaci koji se nalaze u liniji 60. To se desilo zbog naredbe RESTORE. Nakon što je u liniji 10 izvedena naredba READ pokazivač je pokazivao kraj naredbe DATA. Kada je »oraо« naišao na liniju 30 izvršio je naredbu RESTORE koja pokazivač postavlja na početak prve naredbe DATA u programu. Dakle naredbe RESTORE omogućuje nam da podatke koji se nalaze u naredbama DATA čitamo po volji mnogo puta unutar jednog programa.

```
10 READ A$,A
20 PRINT A$;A
30 DATA "ORAO",102
```

Iz ovog primjera vidimo da se string varijable mogu čitati u naredbama READ jednako kao i numeričke varijable. Vidimo da je dozvoljeno u istoj naredbi DATA navesti i tekstualne i brojčane podatke. Važno je jedino da ne pogrješimo u redoslijedu čitanja odnosno da ne pokušamo u numeričku varijablu smjestiti tekstualni podatak. Ako to učinimo računalo će prijaviti grešku. Tekstualni podaci koje navodimo u naredbi DATA mogu se pisati ili s navodnicima ili bez njih. Navodnici su nam potrebni jedino kada uuutar teksta želimo upotrijebiti zarez, jer u protivnom će računalo zarez shvatiti kao kraj teksta.

Povremeno nam se u radu javi potreba da u sredini programe obrišemo sve varijable. Za to nam služi naredba CLEAR. Kada naiđe na ovu naredbu računalo će »zaboraviti« sve vrijednosti pridružene varijablama. To znači da će sve vaujable nakon ove naredbe imati vrijednost nula, a sve string-varijable dužinu nula.

S upotrebom ove naredbe treba biti oprezan jer su vrlo uyatk trenuci kede sa sigurnošću možemo obrisati sve varijable. Na primjer ovu varijablu uije moguće upotrijebiti unutar petlje FOR ... NEXT jer će računalo izgubiti vrijednost kontrolne varijable i prijaviti grešku kada naiđe na naredbu NEXT. Petlju koja je napravljena pomoću naredbe IF naredba CLEAR će najvjerojatnije pretvoriti u beskonačnu petlju.

NUMERIČKE FUNKCIJE

S nekim funkcijama već smo se upoznali (INT, RND). Sada ćemo se pozabaviti ostalim numeričkim funkcijama koje posjeduje »orao«. Da ponovimo: funkcije nisu naredbe i ne mogu samostalno stajati u SASIC-u. Njihova prava namjena je odlučivanje kao prema bilo kojem matematičkom izrazu. Svaka funkcija stoji jedan početak naredbu u zagradi (to je argument funkcije) a funkcija daje određeni rezultat (rezultat funkcije). Sve funkcije koje imaju numerički argument zovu se numeričke i rezultat nazivamo numeričkim funkcijama.

Argument funkcije moramo uvijek navesti u zagradi u protivnom »orao« neće razumjeti izraz koji smo napisali.

Najjednostavnija funkcija u ovoj skupini je funkcije FRE (X). Unutar zagrade kao argument funkcije možemo navesti bilo koju vrijednost. Slične kao i s funkcijom RND ove vrijednost ne utječe na rezultat funkcije. Rezultat ove funkcije je slobodan memorijski prostor u bajtovima koj je preostao u »oriu«. To znači da ako napišemo

```
PRINT FRE (0)
```

»orio« će ispisati koliko slobodnih bajtove imamo u memoriji

Ako imate neki program u memoriji, obrišite ga i pogledajte koliko ma slobodne memorije. Sada upišite bilo koju BASIC liniju i ponovno pogledajte koliko ime slobodne memorije. Kao što je i logično količina slobodne memorije smanjuje se za svakom slijedećom unesenom linijom. Kraćim eksperimentiranjem stećí ćete sliko o tome koliko memorije zauzimaju vaši programi. Vidjet ćete da i nakon vrlo »dugih« programa u memoriji ostaje dosta mjesta. Funkcija FRE poslužit će prilikom razvoja programa, pogotovu u programima koji operiraju s velikim brojem podataka da vidimo kako se troši memorijski prostor. Uz pomoć ove funkcije možemo »izmjeriti« koliko je podataka uneseno u neki program koji radi s podacima. Najprije ćemo napisati cijeli program izmjeriti koliko je memorije preostalo a zatim ćemo u toku izvršenje programa od tog broje oduzeti rezultat funkcije FRE u tom trenutku. Razlika označava memorijski prostor koji su zauzeli podaci

Druga vrlo jednostavne numerička funkcija je funkcija SGN. One nam služi da ispitemo kakav predznak ima neki broj. Ako je argument ove funkcije pozitivan broj one će dati rezultat 1. Za negativan argument rezultat će biti –1 a za nulu 0. Dakle PRINT SGN (5) dat će rezultat jedan PRINT SGN (–5) rezultat minus jedan a nulu ćemo dobiti jednostavno ako je argument 0 PRINT SGN (0)

Svoju primjenu funkcija SGN nalazi u raznim testiranjima. Na primjer nije moguće izračunavati kvadratni korijen iz negativnog broja. U programima u kojima radimo s kvadratnim korijenom možemo upotrijebiti ovu funkciju da testiramo da li je broj negativan

Funkcija ABS ima mnogo šu primjenu od funkcije o kojima smo do sada govorili. Ona izračunava apsolutnu vrijednost broja. Najednostavnije je reći da ove funkcije pretvara predznak broja u plus. To znači ako je argument funkcije ABS negativan broj dobit ćemo isti taj ali pozitivan broj. Funkcije ABS nema nikakav učinak na pozitivne brojeve. Možemo napisati

```
ABS (5) = ABS (-5) = 5
```

Funkcije ABS najčešće nam služi kada radimo s kvadratnim korijenom ili logaritmima. Svaki put kada želimo upotrijebiti neku funkciju koja ne dozvoljave negativan argument, možemo pomoću funkcije ABS učiniti sve argumente prihvatljivima

Jedna od najčešćih operacija u matematičkim formulama je korjenovanje. U velikoj većini potreban nam je kvadratni (drugi) korijen nekog broja. Zbog toga su autori BASICA u BASIC uvratili specijalnu funkciju koja vraća drugi korijen iz argumenta. To je funkcija SQR. O toj funkciji nije potrebno mnogo govoriti, dovoljno je reći da argument funkcije ne smije biti negativan. Upotrebom ove funkcije možemo napisati formulu po kojoj izračunavamo hipotenuzu pravokutnog trokuta iz poznatih kateta (Pitagorin poučak) ovako:

```
C=SQR (A*A+B*B)
```

Oni među vama koji nisu učili trigonometrijske funkcije i logaritme odnosno oni kojima riječi sin cos tan i log ništa ne govore, mogu slobodno preskočiti ostatak ovog poglavlja i odmah prijeći na sljedeće. Ovdje se nećemo baviti objašnjavanjem trigonometrije i logaritmiranja već ćemo proučiti kako se to radi u BASICU.

»Orao« posjeduje funkciju LOG koja daje prirodni logaritam argumenta. Prirodni logaritam je logaritam broja s bazom e (približno 2.71828). Logaritam je dan s nešto manjom točnošću nego što »orao« inače računa. Međutim točnost od pet ispravnih znamenki u potpunosti zadovoljava sve potrebe koje se javljaju za pisanja programa. Argument funkcije LOG mora biti pozitivan broj veći od nule.

Iz prirodnog logaritma možemo izračunati broj (anti logaritmirati) pomoću funkcije EXP. To je funkcija koja u biti izračunava e na x. Ovu funkciju upotrebljavamo u paru s funkcijom LOG ze izračunavanje u kojime su nam potrebni logaritmi.

```
PRINT EXP (1)
```

ispisati će nam vrijednost broja e onako kako je »pamti« orao.

Od trigonometrijskih funkcija »orao« posjeduje funkcije SIN COS TAN i ATN koje odgovaraju sinusu, kosinusu, tangensu i arkus tangensu kuta. Vrijednosti koje se navode kao argument funkcija SIN, COS i TAN kao i rezultat funkcije ATN su u radijanima.

Ako imamo vrijednosti u stupnjevima, možemo ih preračunati u radijane tako de vrijednost u stupnjevima podijelimo sa 180 i pomnožimo sa PI

kut u stupnjevima /180 * PI = kut u radijanima

Ako želimo radijane pretvoriti u stupnjeve, podijelit ćemo vrijednosti sa PI i pomnožiti sa 180

kut u radijanima /PI * 180 = kut u stupnjevima

STRING-FUNKCIJE

Za razliku od numeričkih funkcije string funkcije mogu imati više argumenata. Osim toga string funkcije imaju ili string kao argument ili daju string kao rezultat.

Prve funkcije string kojom ćemo se pozabaviti jest ASC. Argument ove funkcije je string koji ne smije imati dužinu nula. Ove funkcije dat će broj koji je kod prvog karaktera u stringu. Da b-mo to shvatili moremo jednostavno reći kako «oraz» pamti slova i znakove.

Rekli smo već da je memorija mikroračunala podijeljena u bajtove. U jednom bajtu može se nalaziti broj u rasponu od 0 do 255. Svakom slovu u mikroračunalu odgovara jedan broj. Brojevi od 0 do 31 rezervirani su za specijalne znakove. Od 32 do 127 nalaze se slova i znakovi koji postoje u mikroračunalu. Brojevi od 128 do 159 rezervirani su za znakove koje korisnik može sam definirati. Od 160 do 255 nalaze se inverzna slova i znakovi. Svakom znaku pridružen je jedan broj. Na primjer velikom slovu A odgovara broj 65. Funkcija ASC štampat će nam broj koji odgovara prvom znaku u stringu koji je argument funkcije ASC. Pokušajte:

```
PRINT ASC ("A")
```

Sada kad smo shvatili kako «oraz» pamti znakove možemo shvatiti kako usporeduje stringove. Ako u uvjetu unutar naredbe IF napišemo A$>B$ tada «oraz» neće usporediveti dužina stringove kako bi netko pomislio. On će uzeti prvi znak stringa A i prvi znak stringa B te usporediti njihove kodove. Ako su različiti, testiranje će se završiti i eko su isti usporedit će se drugi znakovi u svakom stringu i tako sve do kraja. Slova su poredana po abecedi (slovo A ima kod 65, slovo B 66 i tako dalje), tako da će string koji po abecedi dolazi ze stringe A biti veći od stringe B. Zbog toga što izrazi veći i manji mogu dovesti do zabune u usporedbi stringove umjesto izraze veće uzimamo izraz slijedi, a umjesto izraze manje izraz prethodi. Kažemo da string »AAA« prethodi stringu »BB« odnosno da string »ZZZ« slijedi stringu »EFG«.

U tablici 2. ispisani su svi znakovi koje poznaje mikroračunalo «oraz» i njihovi kodovi. Funkcije ASC dat će broj koji u tablici odgovara pojedinom slovu.

Argument funkcije ASC dakako more biti u zagradi e eko je riječ o tekstu on mora unutar zagrada biti u navodnicima. Nije dozvoljeno tražiti ASC od praznog stringa.

Suprotan efekt funkciji ASC ima funkcija CHR$ Ova funkcija dat će za rezultat string dužine jednog znaka koji sadržava znak čiji je kod bio argument funkcije Na primjer

```
PRINT CHR$ (65)
```

ispisat će na ekranu slovo A. Osim pretvaranja kodove u stringove ova funkcija na mikroračunalu »orao« ima neke mnogo zanimanije primjene. Rekli smo da znakovi čiji su kodovi od 0 do 31 imaju posebnu primjenu ispisivanje nekog od tih znakova može imati utjecaja na izvođenje programe Ove znakove možemo unijeti direktno se tastature pritiskom na tipku (CTL) i neko slovo To možemo učiniti u komandnom kodu Ako želimo upotrijebiti neki od ovih znakova unutar programe poslužit ćemo se funkcijom CHR$ Na primjer ako želimo u programu izvzvati kratki zvučni signal nap set cema

```
PRINT CHR$(7)
```

Ovdje su ispisani neki od najčešće upotrebljavanih kontrolnih znakova i njihov efekt.

4 – pomiče kursor na vrh ekrana bez brisanja ekrana
5 – briše ekran od kursora nadalje
7 – (beep) ispisivanje ovog znaka proizvodi kratak zvučni signal
8 – pomiče kursor za jedno mjesto ulijevo
9 – pomiče kursor za jedno mjesto udesno
10 – pomiče kursor za jedan red dolje
11 – pomiče kursor za jeden red gore
13 – kursor se vraća na početak rede
22 – isključuje i uključuje zvučni signal pri pritisku na tastaturu

Većinu ovih kontrolnih znakova primijenit ćemo kada želimo ljepo urediti ispis na ekranu. Na primjer ako želimo ispisati neku poruku na početku trećeg rede ispisat ćemo prvo CHR$ (12) e zatim dva puta CHR$ (10). Nakon toga kursor će biti na početku trećeg rede. Znakovi CRS (13) i CHR$ (10) automatski se ispisuju nakon svake naredbe PRINT koja nije završila zarezom ili točka zarezom. Prema tome ako napišemo

```
PRINT CHR$(10)
```

kursor će se pomaknuti ze dva reda. Jednom zato što smo to tražili drugi put zato što naredba PRINT nije završene zarezom ili točka zarezom. Ako želimo de u pole naredbe PRINT predemo u novi red stavit ćemo u naredbu PRINT znakove CHR$ (13) i CHR$ (13) kao što je to učinjeno u primjeru

```
10 PRINT CHR$(12);"PRVI RED";CH
R$(10);CHR$(13);"DRUGI RED"
```

Funkciju CHR$ iskoristi-li smo u naslednom programu da ispišemo sva znakove kojima mikroračunala »oraa« raspolaže

```
10 FOR A= 32 TO 255
20 PRINT CHR$ (A);
30 NEXT A
```

Vidimo da kontrolnu varijablu A u petlji mijenjamo u rasponu od 32 do 255 jer smo rekli de znakovi od 0 do 31 služe za posebne funkcije

Naredne funkcije koju često upotrebljavama pri radu sa stringovima je fuuk-ja LEN Upišite sljedeći program

```
10 A$="DUGI STRING"
20 PRINT LEN(A$)
```

Kada pokrenemo ovaj program vidimo na ekranu ispisan broj 11 To je dužina A$ koji je argument funkcije LEN Ova funkcije daje nam broj koji je jednak broju znakove stringa koji je naveden kao ujen argument. Ako u zagradi navodimo tekst umjesto imena stringa tada tej tekst mora biti u navodnicima Navedni znaci ne računaju se u dužinu stringa Fuukcju LEN najčešće upotrebljavamo u kombinaciji sa ostkom od naredbi fuukcija

Funkcije LEFT$ je prva u nizu funkcije koje maju više argumeuata. U fuukcji LEFT$ navodimo dva argumenta od kojih je prvi string e drugi broj Obs argumente navodimo u zagradi e odvejemo ih zarezom Ove funkcije deje ze rezultat string koji je sastavljen od n znakove počevši od lijeve strane stringa koji je naveden kao argument Pogledajmo to na primjeru

```
10 A$="MIKRORAČUNALO"
20 PRINT LEFT$(A$,5)
```

Vidimo da taj program ispisuje riječ MiKRO oduoseno uzima prvih pet slova iz stringa A. Ako uavedemo duzinu koju uzimamo veću od dužina izvornog stringe funkcija LEFT$ uzet će cijeli zvorni string Ako navedemo dužinu manju od 1 računalo će prijaviti grešku. Kao strug-argument funkcije možemo naveati i tekst koji tada mora biti u navodnicima

Funkcije RIGHT$ ima iste argumente kao i funkcija LEFTS i također formira podstring izvornog stringe dužine n znakove jadino što se znakovi ovaj put uzimaju s desne strane izvornog stringa. U sljedećem primjeru

uzeti emo ceam znakove s desne strane izvornog stringa

```
10 A$="MIKRORAČUNALO"
20 PRINT RIGHT$(A$,8)
```

Ako je navedena dužina manja od 1 »orator« će prijaviti greŠku a ako je veća od dužine izvornog stringa uzet će cijeli izvorni string. U svim funkcijama koje operiraju sa stringovima numerički argumenti ne smiju biti veći od 255 jer je i maksimalna dužina stringa ograničena na 255.

Funkcije MID$ ima tri argumenta. Prv argument je string (izvorni string) a druga dva su numerički argument. Ova funkcija daje podatring izvornog stringa počevši od znaka m dužine n. U narednom primjeru iz izvornog stringa izvadit emo podstring koji počinje na šestom znaku i dugačak je pet znakova.

```
10 A$="MIKRORAČUNALO"
20 PRINT MID$(A$,6,5)
```

Prvi numerički argument ne smije biti manji od jedan a ako je veći od dužine izvornog stringa kao rezultat ćemo dobiti prazan string. Drugi numerički argument ne smije biti manji od nule. Ako je on nule dobit ćemo prazan string. Ako je drugi numerički argument veći od preostale dužine izvornog stringe dobit ćemo desni dio izvornog stringa počevši od znaka koji je odredio prvi numerički argumant.

Funkciju MID$ možemo upotrijebiti da pretražimo neki string i ustanovimo da li se u njemu nalazi određeni znak ili neki drugi string. Naredni program analizira string A i traži u njemu prisutnost stringa B. Ako je drugi numerički argument u funkciji MID$ jedan istim programom možamo tražiti prisutnost određenog znaka u stringu.

```
10 A$="MIKRORAČUNALO"
20 B$="ORA"
30 FOR A=1 TO LEN(A$)-LEN(B$)
40 IF B$=MID$(A$,A,LEN(B$)) THE
N PRINT A
50 NEXT A
```

Program nam ispisuje adresu (ili više adresa ako se string pojavljuje više puta) početnog znaka podstringa B u stringu A.

Ponekad je potrebno da broj koji imamo u nekoj numeričkoj varijabli pretvorimo u string. To radimo zato da možemo na neki string pribrojiti (nadovezati) broj ili zato da utječemo na ispis na primjer zbog potpisivanja broja. Pri tome upotrebljavamo funkciju STR$ koja ima

jedan numerički argument. Funkcija STR$ daje za rezultat string koji izgleda onako kako bi izgledao ispis broja na ekranu.

Ako želimo pravilno potpisivati brojeve možemo broj pretvoriti u string zatim izmjeriti «dužinu» broja i pravilno potpisati broj. Kako se to radi prikazano je u narednom programu koji ispisuje brojeve od 1 do 120 sa korakom 9 i pravilno ih potpisuje.

```
10 P$="          "
20 FOR A=1 TO 120 STEP 9
30 A$=STR$(A)
40 PRINT LEFT$(P$,5-LEN(A$));A$
50 NEXT A
```

Potpisivanja smo izvali tako da smo izmjerili dužinu stringa A koji je jednak broju A a zatim ispisali onoliko praznih mjesta ispred broja koliko je manjkalo do broj znajedno a vodećim blenkovima dobija duž nu 5. To smo postigli tako da smo ispisali LEFT$ stringa P dugačak onoliko koliko nam je mjesta manjkalo. Prethodno smo string P definirali kao string koji sadržava blankove. Ovu metodu možemo primjen ti i za previlno potpisivenje decimalnih brojeva jedino što tada moramo mjeriti dužinu samo cjelobrojnog dijela. To ćemo postići tako da prvo izračunamo INT od izvornog broja. Pri potpisivanju negativnih brojeva treba imati u vidu i predznak minus koji će povećati dužinu broja.

Suprotan učinak od funkcije STR$ ima funkcija VAL. One pretvara tekst stringa u numeričku vrijednost. Ako sadržaj stringa mora biti numerički. Dozvoljena je upotrebe znamenki od 0 do 9 i decimalne točke. Ako funkcija VAL naiđe na nenumerički podatak (znak ili slovo) dat će razultat nula. Ako naiđe na numerički podatak pa iza njega nenumerički, uzet će kompletan dio stringa do prvog nenumerčkog podatka i pretvoriti ga u broj. Rezultet će biti nula i ako string ima dužinu nula.

PRINT-FUNKCIJE

U više navrata već smo govorili o naredbi PRINT. Ako pogledate primjere koje smo do sada obrađivali u ovoj knjizi, vidjet ćete da svaki program sadržava ba

rem jednu naredbu PRINT. Da bi mogućnosti u radu s ovom naredbom bile još veće, konstruktori »oria« ugradili su tri funkcije koje rabimo u vezi s naredbom PRINT.

Funkcije POS je zapravo klasične numeričke funkcije. Ime joj je jedan numerički argument koji kao i u funkciji FRE ne utječe na rezultat funkcije. Dakle, u zagradi možemo navesti bilo koji broj. Rezultat ove funkcije je broj koji nam govori na kojem se mjestu (na kojem slovu) u redu nalazi kursor. Prvi znak u redu (početak reda) ima vrijednost nula. Krajnje desni znak u redu ima vrijednost trideset i jedan. Funkcija POS će, dakle, broj u rasponu od 0 do 31. Napišite program:

```
10 PRINT POS(0);
20 GOTO 10
```

Vidimo da nam »orao« ispisuje rastući niz brojeva sve do vrijednosti 31 a zatim počinje iznova. To su brojevi koji pokazuju adresu kursora unutar linije. Ako u našem primjeru izostavimo na kraju linije 10 točke zarez »orao« će stalno ispisivati vrijednost nula. Zašto?

Funkcije SPC služi isključivo unutar naredbe PRINT. Funkcije SPC ima numerički argument i ispisuje onoliko praznih mjesta na ekranu koliko je vrijednost argumenta. Prema tome, ako napišemo:

```
PRINT SPC(20);"A"
```

na ekranu će nam se ispisati slovo A s dvadeset praznih mjesta ispred njega. Ako kao argument navedemo broj koji je veći od 32, ispisivanje će početi u novi red i ostaviti jedan red prazan. Za svakih 32 u argumentu dobit ćemo po jedan prazan red. Na primjer SPC(74) dat će dva prazna reda i deset praznih znakova u sljedećem redu. Argument koji navodimo u funkciji SPC ne smije biti veći od 255 niti manji od nule. Ako kao argument navedemo nulu »orao« će to shvatiti kao da smo napisali 256 i ispisati 256 slobodnih znakova.

Funkciju SPC možemo upotrijebiti i za razdvajanje dva ispisa na ekranu. Tada ćemo ispisati ovu funkciju između dva teksta koje želimo odvojiti.

Druga funkcija s naredbom PRINT je funkcija TAB. U ovoj funkciji navodimo broj koji određuje na kojem će se mjestu početi od početka reda ispisati sljedeći ispis. Na primjer:

```
10 PRINT TAB(10);"A"
```

ispisat će slovo A na desetom mjestu od početka reda. Prvi znak u redu ima broj nula i to je najmanji broj koji možemo navesti kao argument ove funkcije. Najveći dozvoljeni broj je broj 255.

Funkciju TAB ugievnom upotrebljavamo za tabuliranje ispisa pa otud i njeno ime. Naredba

```
10 PRINT TAB(10);"A";TAB(20);"B
"
```

ispisivat će slovo A u desetoj koloni a slovo B u dva desetoj. Ako je druga vrijednost funkcije TAB u istom redu manja od prathodne ova funkcije neće imati nikakav učinak. Ako je argument naveden u funkciji veći od 32 ispis će se nastaviti u novom redu. Isto kao i u funkciji POS. U narednom primjeru primjenjujemo funkciju TAB da pomoću zvjezdice nacrtamo sinusoidu na ekranu

```
10 FOR A=0 TO 6.28 STEP .25
20 B=16+15*SIN(A)
30 PRINT TAB(B);"*"
40 NEXT A
```

Pazite na liniju 20. U njoj izračunavamo vrijednost sinusa. Da bismo je mogli prikazati na ekranu upotrebom funkcije TAB moramo povećati raspon vrijednosti (sinusa nam daje vrijednosti od –1 do +1). Zbog toga množimo vrijednost sinusa sa 15. Time dobivamo vrijednosti u rasponu od –15 do +15. Kako u funkciji TAB nisu dozvoljene negativne vrijednosti moramo na kraju dodati 16 pa dobijemo vrijednosti u rasponu od 1 do 31 koje funkcije TAB nesmetano prihvaća.

O važnost naredbe PRINT osim spomenutih funkcija govori i odluka konstruktora «orla» da nam olakšaju upisivanje ove naredbe. Umjesto da ispisujemo riječ PRINT možemo u svakom trenutku otkucati znak upitnik (?). Sljedeća dva primjera imaju potpuno isti učinak

```
10 PRINT A
xli
10 ? A
```

Ako pri unošenju programa umjesto riječi PRINT napišemo upitnik «orao» će to previšno razumjeti. Ako izlistamo program vidjet ćemo da je upitnik zamijenjen riječju PRINT. Znak upitnika možemo slobodno upotrebljavati pri unošenju programa ali kao pišete programe na papiru pišite riječ PRINT. U protivnom tekstovi neće biti naročito čitljivi.

Sljedeće mogućnost koju nam pruža mikroračunalo «orao» je upisivanje više naredbi u jednoj liniji. Naredbe unutar jedne linije odvajamo dvotočkom (:). Da zvučen je ovaj izraz

```
10 A=5 : B=7 : PRINT A+B
```

Kao što se prilikom normalnog izvođenja programa naredbe izvršavaju odozgo preme dolje tako će se jedna za drugom izvršaveti više naredbi u jednoj liniji slijeva nadesno

Takvo lepisivanje naredbi krije u sebi neke zamke. Kao prvo ovako napisan program je vrlo teško čitati Naredbom GOTO i GOSUB možemo pozvati samo onu naredbu koja ima broj Preme tome ako u liniji 10 imamo tri naredbe na možemo nikako izvesti skok na drugu naredbu u toj liniji Naredba GOTO 10 će izvesti skok na prvu naredbu u liniji 10 i tak nakon što je izvrši, prijeći na drugu naredbu u istoj finji

Iza naredbe GOTO neme svrhe navoditi neke druge naredbe jer se one nikada neće izvršiti Program će kad nađe na naredbu GOTO skočiti u liniju u koju ga ova naredba šalje I neme nečina da se vratimo I izvršimo naredbe iza GOTO Isto tako nema svrhe navoditi naredba iza naredbe REM Kad nađe na ovu naredbu progam ne čita ostatak linije preme tome neće naći ni naredbe koje smo napisali iza naredbe REM

Iza njeći THEN u naredb IF možemo također navesti nekoliko naredbi odvojenih dvotočkama. Treba imati na umu da će se sve ova naredba izvršiti jedino ako je uvjet postavljen u naredbi IF zadovoljen Ako uvjet u naredbi IF nije zadovoljen »cijela« će jednostavno preskočiti ostatak linije U sljedećem primjeru vidimo dosta čestu grešku Petlja FOR NEXT zatvorena je iza riječi THEN u naredbi IF i zbog toga se ne izvršava previno

```
10 FOR A=1 TO 10
20 PRINT A
30 IF A<5 THEN PRINT "MANJE OD
5":NEXT A
```

Vidimo da program radi dok je zadovoljen uvjet u naredbi IF a kada uvjet nije zadovoljen rad programa se prekida Radi toga ovaj program treba napisati ovako

```
10 FOR A=1 TO 10
20 PRINT A
30 IF A<5 THEN PRINT "MANJE OD
5"
40 NEXT A
```

KORISNIČKE FUNKCIJE

U prethodne tri poglavlja proučili smo sve funkcije kojima raspolaže »orao«. Osim navedenih postoji i mogućnost definiranja novih funkcija. Takve funkcije nazivamo korisnički definirane funkcije.

Sami možemo definirati isključivo numeričke funkcije koje imaju jeden argument. Rezultat ovih funkcija je također broj. Svakoj funkciji moramo podijeliti ime dobi je kasnije mogli pozvati. Ime funkcije je jedno slovo iz engleske abecede. Nisu dopuštena naša slova Č Ć Đ Š i Ž. Funkciju definiramo naredbom DEF FN.

Za primjer uzet ćemo definiciju funkcije koje izračunava kvadrat broja.

```
10 DEF FNK (A)=A*A
```

Ovoj funkciji dali smo ime K (to je ono slovo iza riječi FN) i relativni argument A. Ako želimo pozvati ovu funkciju učinit ćemo to ovako

```
20 PRINT FNK (5)
```

Prethodne linije ispisat će nam kvadrat broja 5. U pozivu funkcije možemo kao argument navesti bilo broj bilo varijablu ili matematički izraz.

Rekli smo da funkciji pridjeljujemo relativni argument. Unutar zagrade navodimo ime jedne varijable. To iste ime upotrijebit ćemo s desne strane znaka jednakosti dakle prilikom definiranje funkcije. Kada pozovemo funkciju navest ćemo u zagradi neki drugi podatak (broj ili izraz). Vrijednost tog podatka pridjelit će se relativnoj varijabli navedenoj u funkciji i tada će se izvršiti ono što je navedeno u definiciji funkcije. Ove varijable nema nikakev učinak na ostale varijable i »ze boravi« se odmah nakon što je završeno izračunavanje funkcije. Možemo istodobno u programu imati stvarnu varijablu koje ima isto ime kao i relativne varijable i upotreba relativne varijable neće izmijeniti vrijednost stvarne varijable. Relativne varijable nazivamo još i privremenim ili lokalnim.

Pri definiranju funkcije (izraz s desne strane znaka jednakosti) možemo uzeti i stvarne varijable iz programa. Jedino je ograničenje da ne možemo upotrebljavati

stvarnu varijablu koja se zove isto kao i relativna jer će »oraz« shvatiti da želimo upotrijebiti relativnu.

Funkcija mora biti definirana prije nego što je pozovemo. To znači ako neku funkciju pozivamo u liniji 100 definicija te funkcije mora imati linijski broj manji od 100. Definiranje funkcija ne smije se prilikom izvršenje programe preskočiti naredbom GOTO »Oraz« može iskoristiti funkciju tak nakon što je prešao liniju u kojoj je ona definirana. U praks naredbe DEF FN najčešće smještamo ne sam početak programa.

Unutar jednog programe možemo istu funkciju dve puta definiratr. Prva definicija funkcije primenjivat će se sve dok računalo ne nalde na linju u kojoj se pre definirava funkcije. Nakon ta linija upotrijebit će se druga definicija.

Funkcija ćemo primijeniti kada često unutar jednog programe moramo izračunavati isti izraz.

POLJE PODATAKA

U nakim programima srest ćemo se s većom skupinom podataka koji pripadaju nekoj cjalini. Na primjer moramo izračunati srednju vrijednost temperatura u nekom tjednu. Da bismo to izračunali potrebno nam je sedam podataka koji označavaju temperaturu izmjerenu u svakom pojedinom danu. To možemo napraviti oveko

```
10 READ PO,UT,SR,CT,PE,SU,NE
20 U=PO+UT+SR+CT+PE+SU+NE
30 S=U/7
40 PRINT "SREDNJA TEMPERATURA J
E";S
50 DATA 15,17,21,19,18,20,22
```

Ovaj je pristup moguć kada radimo sa sedam podataka. Zam slite da računate srednju vrijednost tempera ture u god ni dana. Trebalo b nam 365 varijabli! Ovaj problem možemo riješi upotrebom polja podataka. Polje podataka ja skupina podataka kojima je pridru žano zajedničko ime. Svaki takav podatak naziva se element polja i odgovara mu jedan broj odnosno adresa u polju. Možemo reći da su elementi nekog polja ndaksne varijabla.

Da bismo se poslužili poljem moramo naredbom DIM definirati u memoriji prostor za polje. Ime polja formire

se isto kao i ime varijable. Pogledajmo naš primjer rješen upotrebom polja

```
10 DIM A(6)
20 FOR I=0 TO 6
30 READ A(I)
40 S=S+A(I)
50 NEXT I
60 PRINT "SREDNJA VRIJEDNOST JE
";S/7
70 DATA 15,17,21,19,18,20,22
```

Vidimo da smo dobili program koji je nešto duži od programa u prvom primjeru ali ovaj program ostaje isto toliko velik bez obzira koliko velik broj podataka obrađivali. Dovoljno je samo promijeniti vrijednosti u naredbame koje određuju dimenziju polja i broj izvrše nja petlje

Vratimo se naredbi DIM. Naredba DIM u linji 10 de finira polje A koje ima sedam elemenata. To su elementi A(0) A(1) A(2) A(5) i A(6). Svakr od ovih elemenata možemo upotrijebiti isto kao i obične varija ble. Broj u zagradi (indeks) možemo zam jenit varija blom što vidimo u prethodnom primjeru. tada možemo elementima polja manipulirati u petljama. Indeks elemenata polja počinje od nule. Dakle prvi element polja ima indeks nula. Posljednji element polja ima indeks jednak broju koji smo naveli u naredbi DIM. Vidimo da će naredba DIM krenuti polje koje ima jedan element više od broja koji je naveden iza naredbe DIM. Ako nas to zbunjuje, možemo raditi s poljima tako da ne zaboravimo indeks nula, pa će nam polja imati onoliko ele menata koliko smo naveli u naredb DIM.

Ako u programu upotrijebimo indeks koji je manji od nule ili veći od najveće dimenzije polja, računalo će prijaviti grešku. Maksimalna veličina polja, odnosno maksimalni broj elemenata polja određen je jedino raspoloživom slobodnom memorijom.

Sve što smo do sada rekli za numerička polja vrijedi i za string polja. Možemo kreirati string-polje elementi kojih su stringovi s indeksom. Proširimo naš prethodni zadatak da nam ispisuje temperaturu u svakom pojedinom danu. Pri tome ćemo uzeti string polje da lepi šemo dane u tjednu

```
10 DIM A(6) : DIM A$(6)
20 FOR I=0 TO 6
30 READ A(I)
40 NEXT I
50 FOR I=0 TO 6
60 READ A$(I)
70 NEXT I
80 FOR I=0 TO 6
```

```
90 B=B+A(I)
100 PRINT A$(I); TAB(15);A(I)
110 NEXT I
120 PRINT "SREDNJA TEMPERATURA
JE";B/7
130 DATA 15,17,21,19,18,20,22
140 DATA PONEDJELJAK,UTORAK,SRI
JEDA,ČETVRTAK,PETAK,SUBOTA,NEDJ
ELJA
```

Sva polja s kojima smo do sada radili bila su jednodimenzionalna. To znači da je svaki element polja imao samo jedan indeks. Jednodimenzionalna polja možemo zamisliti kao polučiju na kojoj stoje knjige u jednom redu. Prvoj knjizi odgovaraju bi indeks nula, drugoj jedan itd.

Ako uzmemo policu na kojoj knjige stoje u više nivoa, tada za svaku knjigu moramo reći na kojem nivou stoji i koje je knjiga po redu u tom nivou. Vidimo da bi svakoj knjizi odgovarala dva indeksa. Takvo polje nazivamo dvodimenzionalnim poljem. Kasnije ćemo ga tako da u narednoj DIM iza imena polja u zagradama navedemo dva broja odvojena zarezom.

Kada u zagradi navedemo tri broja, dobit ćemo trodimenzionalno polje. U trodimenzionalnom polju svakom elementu polja pridaju tri indeksa. Na primjer, slova unutar jedne knjige možemo gledati kao elemente trodimenzionalnog polja. Svakom slovu odgovaraju tri broja. Prvi broj govori na kojoj se stranici knjige nalazi slovo, drugi broj govori nam red na stranici, a treći broj slova u redu.

Mogu se dimenzionirati i višedimenzionalna polja od četiri, pet ili čak šest dimenzija, ali je pri tome tako krcnano polje u previlu toliko veliko da mikroračunalo ne raspolaže s dovoljno memorije. Na primjer, polje koje ima šest dimenzija i kojemu je svake dimenzija deset, zauzimalo bi oko 5 000 000 bajtova, odnosno oko 5000 K. Vidimo dakle, da je jedino ograničenje prilikom dimenzioniranja polja raspoloživi memorijski prostor.

Dvodimenzionalna i trodimenzionalne polja upotrebljavat ćemo kada imamo dvije skupina istovrsnih podataka pri čemu svakom podatku iz jedne skupine odgovara podatak u drugoj skupini. Tada ćemo upotrijebiti dvodimenzionalno polje kojemu će jedne od dimenzija biti jedan. Uzmimo za primjer da u nekom skladištu treba svakom proizvodu pridružiti jedan broj (skladišni broj) i cijenu. Tada ćemo ze 100 proizvoda upotrijebiti polje dimenzionirano ovako:

```
DIM P(99,1)
```

Tako svakom proizvodu odgovaraju dva elementa polja. Broj prvog proizvoda smjestit ćemo u P(0,0) a cijenu istog proizvoda u P(0,1)

Dvodimenzionalna polja poslužit će nam i pri radu s matricama. Metrice ili tablice su nizovi podataka sastavljeni od jednakih skupina podataka. Ekran na kojem »crao« ispisuje tekst možemo zamisliti kao dvodimenzionalnu matricu u kojoj svakom slovu odgovaraju dva indeksa. Prvi određuje broj reda na ekranu a drugi broj znaka u redu.

Troditmenzionalno polje pomaže u prikazivanju trodimenzionalnih modela. Na primjer: ako želimo označiti neke točke u prostoru svakoj točki odgovaraju tri dimenzije - širina, dužina i visina.

Polje s više od tri dimenzije vrlo se rijetko primjenjuju. Osim toga čovjek prilično teško radi s tim pojmom jer pri radu ne može zamisliti neki prihvatljivi oblik organizacije podataka. To je zbog toga što smo u svakodnevnom životu navikli raditi s modelima koji imaju najviše tri dimenzije.

LOGIČKE OPERACIJE

Svi brojevi i znakovi kojima mikroračunalo barata smješteni su u bajtove. U bajtu može biti vrijednost od 0 do 255. Pogledajmo zašto baš ove vrijednost.

Mikroračunalo sve podatke kojima barata i sve operacije koje izvršave prikazuje u obliku električnih impulsa. Električni impulsi mogu biti u dva različita stanja (ima struje ili nema struje). Odnosno postoji protok električne struje kroz neki sklop (sklop vodi) ili ne postoji (sklop ne vodi). Stanje u kojem određeni sklop ne vodi označavamo nulom, a stanje u kojem taj sklop vodi jedinicom. Ova stanja nazivamo i logička stanja sklopa pa kažemo logička nula ili stanje logičke nule i logička jedinica. Takav sklop može zabilježiti samo opisana dva stanja i takav sklop nazivamo BIT. Dakle, jedan bit ima stanje 1 ili stanje 0.

Ako upotrijebimo dva bita možemo prikazati četiri različita stanja. To su

- oba bita su 0
- prvi bit je 0, drugi 1
- prvi b t je 1 drugi 0
- oba bita su 1

Vidimo da sa dva bita možemo prikazati različite vrijednosti. Takvo prikazivanje brojeva naziva se binarni prikaz brojeva ili prikaz brojeva s bazom dva.

DEC	BIN	HEX
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

U tablici vidimo binarni prikaz svih brojeva od 0 do 15. Vidimo da smo prikazujući vrijednosti od 0 do 15 iscrpili sve moguće kombinacije binarnih brojeva sa četiri znamenke. Možemo reći da sa četiri binarne znamenke možemo prikazati 16 različitih stanja. Moguće je metematički dokazati da četiri binarne znamenke čine dve ne četvrtu različitih stanja, odnosno da n binarnih znamenki čine dva na n različitih stanja. Svakoj vrijednosti u prethodnoj tablici pridružili smo jedan broj ili slovo što vidimo u trećem stupcu tablice. Vrijednostima od 0 do 9 pridružili smo brojeve od 0 do 9, a vrijednostima od 10 do 15 slova od A do F. Tako svakoj vrijednosti od 0 do 15 pripada jedan znak. Takav način prikazivanja brojeva nazivamo heksadecimalnim sistemom prikazivanja brojeva. On ima bazu 16. Pomoću jedne heksadecimalne znamenke možemo prikazati četiri binarne znamenke.

Jedan bajt odnosno jedna osnovna jedinica memorije sastoji se od osam binarnih znamenki. Dakle, svaki bajt možemo prikazati pomoću osam jedinica ili nula. Ako otac« u nekom bajtu ima vrijednost 130, on će u tom bajtu imati sljedeći broj

10000010

Se osam binarnih znamenki možemo prikazati dve na osmu različitih stanja, odnosno 256 različitih vrijednosti. Kako je jedne od vrijednosti nula, najveći broj koji se može nalaziti u jednom bajtu je 255.

Binarno prikazivanje brojeva je za ljude jako nepregledno i teško čitljivo. Zbog toga stanje pojedinog bajta prikazujemo pomoću dva heksadecimalna znaka. Pretvorbe iz binarnog u heksadecimalno je vrlo jednostavne. Dovoljno je da uzmemo lijeve četiri bita iz binarnog broja i umjesto njih napišemo odgovarajuću heksadecimalnu vrijednost iz tablice. Sada to isto učinimo s desna četiri bita i dobili smo dvoznamenkasti heksadecimalni broj koji prikazuje stanje određenog bajta. Na primjer

```
11111111 = FF
10100101 = A5
00011110 = 1E
```

Ako se u heksadec mal nom broju pojavljuje neke vri jednost označena slovom onda nam je odmah jasno da je u pitanju heksadecimalni broj Za broj 56 ne znamo da li je riječ o decimalnom broju 56 ili heksade cimalnom broju Zbog toga lepred heksadecimalnih brojeva obično pišemo malo slovo h ir znak & Broj 56 heksadecimalno napisat ćemo h56 ili &56

Na kraju ove knjige imate program koji pretvara bro jeve iz binarnih u decimalne ili heksadecimalne z de cimalnih u binarne ili heksadecimalne i iz heksadeci malnih u decimalne ii binarne

Bez obzira ne to kako je prikazan pojedini broj sve matematičke operacije davat će točan rezultat Š biner nim brojev ma možemo izvodit određene logičke ope racije To su logičke operacije NOT (ne) OR (ili) i AND (i)

Logičke operacija NOT izvsi će inverziju ili negaciju bita Prema tome NOT 1 jednako je nula, a NOT 0 jednako je jedan To obično prikazujemo ovako

A	NOT A
0	1
1	0

Ako izvedemo operaciju NOT nad nekim brojem svi bitovi tog broja promijenit će vrijednost Sve nule postat će jedan e sve jedinice nule Ne primjer

```
NOT 11001010 = 00110101
```

Logička operacija OR izvodi se između dva broja. Ako izvedemo OR između dva bita rezultat će biti jeden ako je bilo koji od bitove jeden Prema tome logička operacija OR daje vrijednost nula jed no ako su oba bita nula U tablici to izgleda ovako

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Kada izvedemo logičku operaciju OR između dva osmob tna broja dobit ćemo osambitni broj u kojem su jedinice svi onr bitovi od kojih je berem jeden bil u izvornim podacima 1 Dakle

```
11001010 OR 01010101 = 11011111
11110000 OR 10101010 = 11111010
```

Ako izvedemo OR između nekog broja i nule rezultat će biti isti kao prvi broj a ako izvedemo OR između nekog broja i hFF (255) rezultat će biti hFF

Logička operacija AND također se izvodi između dva broja i daje rezultat 1 jedino ako su oba broja 1. To u tablici izgleda ovako

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Logičko AND između dva osambitna broja dat će osambitni broj u kojemu su jedinice oni bitovi koji su u oba izvorna broja jedan

```
11001010 AND 01010101 = 01000000
11110000 AND 10101010 = 10100000
```

Logičko AND između broja A i nule dat će rezultat nule a logičko AND između broja A i hFF dat će rezultat A

Osim ovih osnovnih logičkih operacija upotrebljava se još i složena log čka operacija XOR ili EOR. Ovu operaciju nazivamo ekskluzivno ili, a možemo je dobiti prema svoj formuli

```
A XOR B =
A AND NOT B OR NOT A AND B
```

Ova operacija primijenjena na dva binarna broja daje jedan jedino ako je samo jedan od ta dva broja jedan. U tablici to izgleda ovako

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Logičke XOR između dva osambitna broja dat će osambitni broj u kojem su jedinice oni bitovi u kojima je samo jedan od izvornih bitova 1

```
11001010 XOR 01010101 = 10011111
11110000 XOR 10101010 = 01011010
```

XOR broja A i nula dat će broj A a XOR broja A i hFF dat će NOT A

```
PRINT 10 AND 3
```

Ispisat će broj 2 jer te u binarnom obliku izgleda ovako

```
00001010 AND 00000011 = 00000010
```

LOGIČKE OPERACIJE U BASICU

«Orlov» BASIC posjeduje logičke operacije NOT OR i AND Ove operacije možemo izvoditi na brojevima ali pri tome moramo biti svjesni da su u pitanju binarne logičke operacije dakle izvode se na binarnim prikazima brojeva s kojima radimo

Na primjer

```
PRINT 10 OR 3
```

Ispisat će broj 11 jer to u binarnom obliku izgleda ovako

```
00001010 OR 00000011 = 00001011
```

Vrlo rijetko će se javiti u programu potrebe da logičke operacije izvodimo na brojevima Mnogo češće primjene ovih operacija je povezivanje uvjeta Unutar naredbe IF možemo navesti nekoliko uvjeta povezanih logičkim operatorima Ako dva uvjeta povežemo operatorom naredbe AND iza nječi THEN izvast će se jedino ako su ispunjena oba postavljena uvjeta Na primjer

```
IF A=5 AND B>3 THEN PRINT "UVJET
 ISPUNJEN"
```

Ispisat će UVJET ISPUNJEN jedino ako je A jednako 5 i B veće od 3 Ako bilo koji od ova dva uvjeta nije ispunjen, neće biti isp san tekst.

Povežemo li dva uvjeta logičkim operatorom OR izvest će se naredba iza riječi THEN ako je barem jedan od postavljenih uvjeta ispunjen. Na primjer

```
IF A=5 OR B>3 THEN PRINT "UVJET ISPUNJEN"
```

ispisat će UVJET ISPUNJEN ako je A jednako 5 ili B veće od 3 ili ako su ispunjena oba uvjeta. Jedino ako nije ispunjen niti jedan od ova dva uvjeta, neće biti izvršena naredba PRINT.

Možemo upotrijebiti i operator NOT pa će naredba

```
IF NOT A=5 THEN PRINT "UVJET ISP UNJEN"
```

ispisati UVJET ISPUNJEN ako A NIJE jednako 5. Vidimo da operator NOT pretvara izraz jednako u razl ično. Učinak operatora NOT na pojedine uvjete je ovakav

uvjet	NOT uvjet
=	<>
<	>=
>	<=
<=	>
>=	<
<>	=

Dopušteno je i unošenje složenih izraza. Na primjer

```
IF A=5 AND B>3 OR C=0 THEN PRIN T "UVJET ISPUNJEN"
```

Logičke operacije imaju prioritete kao i numeričke operacije. Najviši prioritet ima logička operacija NOT, zatim slijedi AND a najniži prioritet ima logička operacija OR.

Prethodni primjer analizirat će se ovako: najprije će se izvesti log ičko AND dakle, ispitat će se da li su oba uvjeta A jednako 5 i B veće od 3 ispunjena a zatim će se između tog zaključka i uvjeta C jednako 0 izvršiti logičko OR. Dakle u prethodnom primjeru tekst UVJET ISPUNJEN ispisat će se ako je A=5 i B>3 ili ako je C=0 ili ako su svi uvjeti ispunjeni.

U kreiranju složenih logičkih uvjeta dopuštena je upotreba zagrada i pomoću njih izmjena redoslijeda operacija. Na primjer

```
IF A=5 AND (B>3 OR C=0) THEN PRI NT "UVJET ISPUNJEN"
```

Tekst UVJET ISPUNJEN ispisat će se ako je A jednako 5 i barem jedan od uvjeta u zagradi ispunjen

Primjena složenih uvjeta znatno nam olakšava program ranje i smanjuje potreban broj naredbi IF u programu Međutim složeni logički uvjeti su zbog svoje zamršenosti vrlo cest uzrok grašake u programu Zbog toga početnik ne bi trebair prstjerivati u sklapanju složenih logičkih uvjeta.

GRAFIKA

Jedno od najčešćih upotreba računala u suvremenom životu je izrada grafike Čovjeku je mnogo lakše da stiglede odnose između nekih veličine ako su one prikazane grafičke Time se informacija do čovjeka brže i lakše prenosi što je osobito važno za poslove u kojime treba donositi mnogo odluka u vrlo kratkom roku

» Orao« raspolaže mogućnošću crtenje grafike Za potrebe rada na grafici ekran moramo shvatiti kao prvi kvadrant koordinatnog sustava Donja lijeva točka ekrana ima koordinate 0 0 a gornje desna koordinate 255 255 Vidimo da je ekran kvadratna ploča s 256 x 256 točkica Bilo koju od ovih točaka možemo »osvijetliti« Za rad s grafikom služit će nam tri BASIC instrukcije

Instrukcija PLOT iza koje slijede dva broja osvjetlit će na ekranu točku koju adresiraju ta dva broja Napišite

```
10 PRINT CHR$(12)
20 PLOT 127,127
```

Vrlimo da se pojavila bijela točkica na sredini ekrana. Prvi broj označava koordinatu x odnosno za koliko u desno pom čemo točku koju crtamo Drugi broj je koordinate y odnosno na kojoj visini crtemo točkicu Prema tome 0 255 adresira gornju lijevu a 255 0 donju desnu točku ekrana

Instrukcija DRAW povlači liniju od grafičkog kursora do adrese koju odreduju brojevi navedeni iza naredbe DRAW

Proučimo prvo što je grafički kursor Kao što postoji kursor za tekst koj odreduje kamo će ići naredni upisani znak tako postoji i grafički kursor Kada uključimo računalo grafički kursor nalazr se u donjem lijevom kutu ekrana odnosno na adresi 0 0 Svake grafičke naredbe koju izvršimo pomaknut će grafički kursor. Instrukcija PLOT postavlja grafički kursor na ono mje-

sto na ekranu na kojem nacrta točku. Instrukcija DRAW smješta grafički kursor u točku u kojoj je završeno crtanje.

Brojevi koje navodimo iza naredbe DRAW isti su kao i u naredbi PLOT - samo što ovdje određuju gdje će završiti i nija. Prema tome, ako napišemo

```
10 PLOT 0,0
20 DRAW 255,255
```

iscrtat ćemo dijagonalu ekrana. Linija 10 postavlja grafički kursor u donji lijevi kut ekrana, a linija 20 povlači liniju od grafičkog kursora do gornjeg desnog kuta ekrana. Upišite program

```
10 FOR A=0 TO 255 STEP 8
20 PLOT A,0
30 DRAW A,255
40 PLOT 0,A
50 DRAW 255,A
60 NEXT A
```

Vidimo da ovaj program iscrtava kvadratnu mrežu. Promjenom vrijednosti iza riječi STEP u liniji 10 možamo dobiti mrežu različite gustoće.

Instrukcija MOVE služi za pomicanje grafičkog kursora. Zapravo ova instrukcija ima potpuno isti učinak kao i instrukcija PLOT jedino što na crta točku. Prema tome, u prethodnom primjeru mogli smo umjesto instrukcija PLOT staviti instrukciju MOVE

Vidjeli smo u prethodnom primjeru da »oras« crta bijelom bojom po crnoj podlozi. Ako želimo »obrisati« nacrtano odnosno crtati crnom bojom upotrijebit ćemo naredbu MODE. Ako napišemo MODE 1 »oras« će crtati crnom bojom. Ako napišemo MODE 128 crtat će se crnom bojom na bijelој podlozi a bijelom bojom na crnoj. Kažemo da se ovdje crta invertiranjem. Pomoću naredba MODE 0 vratit ćemo način crtanja kakav je bio na početku. Pri svakom resetiranju ili gašenju mikroračunala postavit će se vrijednost MODE 0.

Kombinirajući ove načina crtanja uz pravilnu upotrebu instrukcija za crtanje moguće je na »oru« dobiti zaista impresivne grafike. Grafička rezolucija od 256 x 256 točaka zadovoljava u velikoj većini primjera grafika na mikroračunalu. Naredni program crta grafikon funkcija sinus i kosinus u rasponu od 0 do 4*PI upotrebom instrukcije PLOT

```
10 PRINT CHR$(12)
20 MOVE 0,127
30 DRAW 255,127
40 FOR I=0 TO 255
50 A=127+126*SIN(6.28/128*A)
```

```
60 PLOT I,A
70 A=127+126*COS(6.28/128*A)
80 PLOT I,A
90 NEXT I
```

Vidimo da su grafikoni prikazani pomoću niza točkica. Bolje rješenje dobit ćemo ako upotrijebimo instrukciju DRAW međutim tada moramo crtati jedan po jedan grafikon zbog toga što upotrebljavamo položaj grafičkog kursora. Zamijenite u prethodnom primjeru instrukcije PLOT instrukcijama DRAW i vidjet ćete da nije moguće crtati dva grafikona odjednom. Zbog toga u narednom primjeru svaki grafikon ima svoju petlju

```
10 PRINT CHR$(12)
20 MOVE 255,127
30 DRAW 0,127
40 FOR I=0 TO 255
50 A=127+126*SIN(6.28/128*A)
60 DRAW I,A
70 NEXT I
80 MOV 0,253
90 FOR I=0 TO 255
100 A=127+126*COS(6.28/128*A)
110 DRAW I,A
120 NEXT I
```

Vidimo da su tako nacrtane krivulje znatno homogenije jer ih instrukcija DRAW pravilno povezuje. Pogledajte liniju 80 u kojoj smo zbog crtanja drugog grafikona prvo postavili grafički kursor na početak grafikona kosi nula funkcije. Unutar svake linije DRAW u programu navodili smo dva parametra. Ti parametri određuju koordinate točke na kojoj će završiti iscrtavanje linije (početak je određen grafičkim kursorom). Unutar jedne linije DRAW možemo navesti i više parametara. Broj parametara mora biti paran (2 4 6 itd). Na primjer ovaj program nacrtat će na ekranu kvadrat

```
10 MOVE 50,50
20 DRAW 150,50,150,150,50,150,5
0,50
```

Broj parametara koja navedemo unutar linije DRAW ograničen je jedino maksimalno dozvoljenom dužinom linija (72 znaka).

Takvom upotrebom linije DRAW možemo dobiti razne mnogokute.

Naredba DRAW neće nam pomoći pri crtanju kruga (trebalo bi nam nekoliko stotina parametara). Zbog toga su autori BASICA uključili naredbu CIR iza ove naredbe dolaze tri parametra. Prva dva određuju koordinate središta kružnice. Treći parametar određuje polumjer kružnice. Ako je treći parametar veći od 128 »crao« će kao polumjer kružnice uzeti razliku između

broja 255 i navedenog polumjera. Na primjer ako navedemo naredbu

```
CIR 128,128,200
```

»orao« će nacrtati na sredini ekrana krug čiji je polumjer 55 (255-200). Prema tome naredba

```
CIR 128,128,56
```

dat će potpuno isti rezultat kao i u prvom slučaju. Ovaj program crta niz koncentričnih krugova

```
10 FOR I=10 TO 120 STEP 10
20 CIR 128,128,I
30 NEXT I
```

Vidjeli smo čitav niz naredbi za rad s grafikom. Osim navedenih naredbi postoji i jedna funkcija koja služi u radu s grafikom. Njena je specifičnost da se uz nju ne navodi parametar. Funkcija DOT omogućuje nam da »pročitamo« stanje neke točke na ekranu. Ako je točka koju čitamo »osvijetljena« dobit ćemo rezultat 1, a ako je »neosvijetljena« dobit ćemo rezultat 0. Točke čitamo ovom naredbom

```
A=DOT
```

Prema tome ovu funkciju možemo upotrijebiti jedino tako da nekoj varijabli pridružimo vrijednost (izraz DOT ne možemo unijeti unutar naredbe PRINT). Funkcija DOT čita stanje točke koju određuje položaj grafičkog kursora. Prema tome ovaj program ispisuje nam stanje točke na sredini 128 128

```
10 MOVE 128,128
20 A=DOT
30 PRINT A
```

JOŠ JEDNOM O EKRANU

Dosad smo u više navrata govorili o ekranu i o naredbi PRINT. Ako pažljivo proanaliziramo BASIC mikroračunala «orao» vidjet ćemo da se najveći broj naredbi i funkcija odnosi upravo na rad s ekranom. To je zbog toga što svaki program mora ispisati rezultat svog izvođenja, a što je rezultat lijepše ispisan to ćemo lakše primiti informaciju koju nam računalo prenosi.

Vrlo često je u programu potrebno obrisati ekran. Pažljivim čitanjem vjerojatno ste vidjeli da se ekran može obrisati ispisivanjem kontrolnog znaka 12. Mnogo lijepše je za brisanje ekrana uzeti naredbu CLS. Naredba CLS ima potpuno istu funkciju kao i kontrolni znak 12 dakle briše ekran i postavlja kursor u gornji lijevi kut ekrana. Treba imati na umu da naredbe CLS ne utječe na položaj grafičkog kursora.

U poglavlju u kojem smo govorili o kontrolnim znakovima, vidjeli smo da upotrebom kontrolnih znakova možemo pomicati kursor po ekranu. Mnogo je lakše upotrebom naredbe CUR postaviti kursor na željeno mjesto. Iza naredbe CUR dolaze dva parametra. Prvi broj određuje položaj kursora u redu a drugi broj red na koji smještamo kursor. Gornji lijevi kut ekrana ima koordinate 0 0 gornji desni 31 0 donj desni 31, 31. Evo programa s naredbom CUR da lijepše dijagonalu ekrana uz pomoć slova X.

```
10 FOR I=0 TO 31
20 CUR I,I
30 PRINT "X";
40 NEXT I
```

Inverzni ispis na ekranu uključujemo pomoću naredbe INV iza ove naredbe slijedi jedan parametar koji može biti 0 ili 1. 0 isključuje inverzni ispis (normalni ispis) a 1 uključuje inverzni ispisivanje slova.

```
10 INV 1
20 PRINT "INVERZNO"
30 INV 0
40 PRINT "NORMALNO"
```

Kada smo govorili o grafici rekli smo da pomoću funkcije DOT možemo pročitati stanja neke točke na ekranu. «Orao» posjeduje sličnu funkciju za čitanje slova s ekrana. Kao i uz funkciju DOT ne navodimo nikakve parametre. Izraz

A$=SCREEN$

pročitat će slovo s ekrana i pridijeliti ga string-varijabli

A. Koje će se slovo čitati, određuje položaj kursora. Prama tome možemo upotrebom naredbe CUR I funkcija SCREEN$ pročitati željeno područja na ekranu. Ako ne mjestu koja čitamo nema prepoznatljivog znaka (na tom mjestu nalazi se nekakva grafika ili predefinirana slova) string-varijabla A tmat će dužinu 0. Prama tome neće joj biti pridijeljena nikakva vrijednost.

Vrlo često kada kombiniramo grafiku i tekst na ekranu za takst predvidimo samo određeni prostor na ekranu. Takav prostor u računskoj terminologiji naziva se prozor. Mikroračunalo «orao» ima mogućnost definiranja jednog tekstualnog prozora. Definira se naredbom VDU iza koja slijeda četiri parametra. Parametri mogu biti u rasponu od 0 do 31. Prvi parametar određuje ljevi rub prozora, drugi određuje desni rub treć parametar definira gornji a četvrti donji rub prozora. Logično ja da drugi parametar mora biti veći od prvog odnosno da četvrti mora biti veći od trećeg. Ako naredbu VDU navedemo baz parametra poništit ćemo defin ciju prozora i u isto vrijema obrisati ekran. Možemo radi da VDU baz parametra znači isto što i niz VDU 0 31 0 31 CLS.

Vidjali smo da tekst na ekranu možemo istaknuti tako da ga ispišemo inverzno. Dosta često potrebno je da na ekranu ispišemo neku informaciju krupnijim slovima. To osobito dolazi do izražaja ako na računalu pišamo neku informaciju koja će se čitati s mesto veće udaljenosti (npr. na izložbama prirodbama i at). Za ispis vecih slova na mikroračunalu «orao» ima naredbu LETTER iza koja slijede tri parametra. Prvi parametar ja tekst koji želimo ispisati drugi parametar određuja veličinu slova u slovnim mjestima (npr. 3 znači slovo v soko tri normalna slova). Treći parametar mora biti u rasponu od 0 do 3 i ima ovo značenje.

0 – ispiše tljeva na desno
1 – ispis odozdo prema gore
2 – ispis sdesna nalijevo
3 – ispis odozgo prema dolja

Mjesto na koje će se takst ispisati određujemo grafičkim kursorom. Pomoću naredba MOVE postavit ćemo grafički kursor na mjesto gdje želimo da nam buda donji ljevi kut prvog slova. Nas primjer ispisuja slove A preko cijeloj ekrana.

```
10 MOVE 0,0
20 LETTER "A",32,0
```

Naredbu LETTER iskoristit smo i u narednom programu tako da riječ «orao» ispišemo šest puta slovima po rubu ekrana.

```
10 CLS
20 FOR I=0 TO 3
30 READ A,B
40 MOVE A,B
50 LETTER "ORAO",6,I
60 NEXT I
70 DATA 0,0,255,0,255,255,0,255
```

MEMORIJA

U nekoliko navrate rekao sam da je memorija podijeljena na bajtove. Svaki od ovih bajtova ima svoju adresu, odnosno jedan broj koji mu je pridijeljen i pomoću kojeg možemo pročitati ili upisati vrijednost u taj bajt. Procesor 6502 koji je ugrađen u mikroračunalo »oraO« adresu bajta čuva u dva bajta. Prema tome, adresa pojedinog bajta jest binarni broj od šesnaest znamenki. Iz formule dve na šesnaestu možemo zaključiti da mikroprocesor 65#2 može adresirati 65536 bajtova, odnosno 64*1#24 bajta ili 64K. Neki od ovih bajtova nalaze se u području memorije čiju vrijednost ne možemo mijenjati (ROM) a ostali su ili neupotrijebljeni ili pokriveni područjem memorije koje možemo mijenjati. Ne postoji nikakve opasnosti da oštetimo računalo ni onda kada pokušamo promijeniti nešto u ROMU ili na adresi na kojoj nema memorije.

Promjenu adresene memorijske lokacije postižemo upotrebom naredbe POKE. Iza ove naredbe slijede dva broja. Prvi označava adresu bajta u koji smještamo vrijednost a drugi vrijednost koja se smješta u taj bajt. Prvi broj more biti u rasponu od 0 do 65535 a drugi od 0 do 255. Ako je bilo koji od ovih brojeva izvan predviđenog opsega, računalo će prijaviti grešku.

Određene memorijske lokacije sadržavaju vrijednosti koje su od vitalnog značenja za uspješan rad BASICA. To su lokacije na adresama od 0 do 511 i njihov sadržaj nije preporučljivo mijenjati osim ako točno znamo što radimo. Na adresama iznad 1024 počinje BASIC-program i u tom području mijenjanje vrijednosti pojedinih brojeva može izazvati promjenu u BASICU što će kasnije dovesti do neispravnosti u radu programa. Ako želimo eksperimentirati s mijenjanjem sadržaja pojedinih memorijskih adresa najbolje je da radimo pri vrhu memorije određenu za BASIC-program pod pretpostavkom da nam BASIC ne dolazi do te lokacije.

Prilikom uključivanja računala »oraO« nas pita

```
MEM SIZE  ?
```

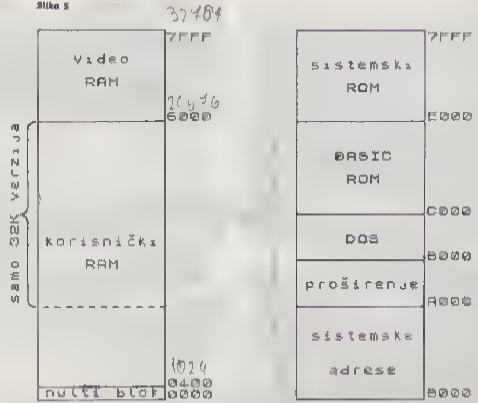
i tada možemo odgovoriti koliko memorijskog prostora želimo rezervirati izvan područje dohvatljivog BASICU. Tako možemo rezervirati prostor na kojem će biti naši podaci ili strojne rutine sigurni od opasnosti da ih prebriše BASIC.

Sadržaj pojedine memorijske lokacije možemo pročitati funkcijom PEEK. Za ovu funkciju vrijedi sve ono što je rečeno za numeričke funkcije. Argument ove funkcije koji se navodi u zagradi iza riječi PEEK označava adresu bajta koji čitamo i mora biti u rasponu od 0 do 65535. Naredni

```
PRINT PEEK (1000)
```

ispisat će sadržaj bajta koji ima adresu 1000. Za ek-

Slika 5


32767
7FFF
Video
RAM
24576
6000
samo 32K verzija
korisnički
RAM
1024
0400
0000
nulti blok
7FFF
sistemska
ROM
E000
BASIC
ROM
C000
DOS
B000
proširenje
A000
sistemske
adrese
9000

parmantiranje s naredbama PEEK i POKE upotrijebit ćemo video RAM. To je područje u memoriji u kojem »oko« čuva sliku. Ako izmijenimo bilo koju vrijednost u ovom području, to će odmah rezultirati promjenom na ekranu. Na primjer

```
POKE 24607,255
```

crta crticu u gornjem desnom kutu ekrana. Svakoj točki ekrana odgovara jedan bit u određenom bajtu. Osam točkica ekrana smješteno je u jeden bajt. Kada je bit koji pripada određenoj točkici postavljen na vrijednost 0 točkica je crne boje. Ako je vrijednost bita postavljena na 1 točkica je bijele boje. Kada u jeden bajt memorije ekrana smjestimo vrijednost 255 koja binarno izgleda ovako

```
11111111
```

upale se sve točkice koja odgovaraju tom bajtu. Zbog toga je u prethodnom primjeru nestala crtica.

Videomemorija u mikroračunalu »oreo« nalazi se na adresama od 24576 do 32767 i zauzima 8K. Ako pomnožimo 256 sa 256 (koliko točkica ima na ekranu) i to podijelimo sa 8 (jer osam točaka zauzima jedan bajt) dobit ćemo točno 8K. Unesite ovaj primjer

```
10 FOR A=24576 TO 32768
20 POKE A,255
30 NEXT A
```

Vidimo da se slike popunjava odozgo prema dolje. U videomemoriji prvi bajt označava prvih osam točaka gornjega lijevog kuta ekrane. Zatim slijedi naredna osam točaka i tako dalje sve dok se ne završi prvi red točkica na ekranu. Svakom redu točkica odgovara 32 bajta (32*8 = 256). Nekon toga si sljeda 32 bajta za drugi red točkica i tako dalje sve do završetka ekrana.

Instrukciju POKE možemo skoristi ti de definiramo vlastite znakove. Za potrebu definicije znakova »oreo« posjeduje određene bajtove u kojima cuve vrijednost nule. Ako promijenimo vrijednost u tom bajtu »oreo« će to shvatiti kao da želimo predefinirati slova ili znakove. S obzirom na to de jedan bajt nije dovoljan de se čuva adresa nekog bajta u memoriji, »oreo« će prilpo stavit da je vrijednost koju mu demo prvi bajt adrese a da je drugi bajt nula. Time smo ograničeni de moremo definiraju naših znakova stavit na adresu koje u binarnom obliku ime posljednjih osam znamenki nule. Dio memorije koji počinje od adrese koja u binarnom obliku završava sa osam nula i dug je 256 bajtova naziva se stranica (page) memorije. Zbog toga kažemo da se definicije naših znakove mora nalazi ti u jednoj stran c memor je.

Uzet ćemo za primjer stran cu memorije koja počinje na adresi 24064. Prvi bajt adrese za tu stranicu je 94

(sve vrijednosti dane su decimalno). Sada ćemo na tu adresu smjestiti osam bajtova koji će označavati definiciju znaka. Svakoj točki našeg znaka odgovara jedan bit u bajtu. Ako želimo da ta točka svijetli kada se znak ispiše postavit ćemo u bit vrijednost jedan. Predefinirati možemo samo 32 znaka odjednom. Zbog toga ćemo prvo u liniji ma 10 do 30 postaviti u sva bajtove vrijednost 255. Time će sv znakovi biti predefinirani u bijeli kvadratić. Sada možemo pristupiti definiciji našeg znaka. Definirat ćemo crni kvadratić s bijelim rubom. Naš znak možemo zamisliti kao matricu od 8 x 8 točkica. Gornji red točaka je prv bajt u definiciji drugi red točaka je drugi red i tako dalje. U prvom bajtu želimo da nam svijetle sve točke pa ćemo u njega smjestiti vrijednost 255 (11111111 binarno). U drugom redu želimo da nam svijetle prva i posljednja točkica, pa ćemo staviti vrijednost 129 (10000001 binarno). Treći četvrti peti šesti i sedmi red su su kao i drugi pa u njih ide vrijednost 129. U posljednjem redu ponovno želimo da svijetle sve točkice i zato stavljamo vrijednost 255.

Sada smo smjestili definiciju znaka u memoriju i još samo treba da kažemo «ru» koji dio znakova želimo predefinirati. Svakom dijelu znakova odgovara jedan bajt. U slijedeći tablici vidite kojem dijelu slova odgovara koji bajt.

512 — kerneliški definirani znakovi
513 — znakovi i brojevi
514 — velika slova
515 — mala slova

Predefinirat ćemo mala slova. Definiciju znakova smjestili smo na stranicu broj 24 i zato ćemo u bajt 515 koji odgovara malim slovima smjestiti vrijednost 94. Evo programa koji sve to obavlja

```
10 FOR I=24064 TO 24319
20 POKE I,255
30 NEXT I
40 FOR I=24064 TO 24071
50 READ A
60 POKE I,A
70 NEXT I
80 POKE 515,94
90 DATA 255,129,129,129,129,129
,129,255
```

Prvi znak u setu malih slova je strelica gore (posljednja tipka u drugom redu tastature). Pritiskom na tu tipku dobit ćemo znak koji smo definirali. Ako uključimo mala slova (tipka (FF1) v d mo da bez obzira na to koja slovo pritiskamo dobivamo bijeli kvadratić.

Mala slova možemo «vratiti» ako u adresu 515 vratimo vrijednost nula (POKE 515,0) ili ako resetiramo računalo.

Vidjeli smo u prethodnoj tablici da adresa 512 odgovara kerneliški definiranim znac ma. To su znaci koji odgovaraju kodovima od 128 do 159. Ako nam unutar

nekog programa trebalu korisnički znaci a željimo zadržati sva slova i znakove možemo definirati to područje. Jedini problem je što ovako kreirane znakove ne možemo unositi direktno s tastature, već ih moramo ispisivati pomoću funkcije CHR$

Ako predefiniramo mala slova, velika slova ili znakove možemo ih ispisivati inverzno pritiskom na tipku (PF3)

Možemo predefinirati sve znakove, na primjer »oraz« može raditi u čitlici. Jedini je problem što gubimo mogućnost rada u ekranskom editoru. Kad pritisnemo tipku (PF4) »orso« će prepoznati znak preko kojeg prelazi kursor jedino ako taj znak točno odgovara slovima koje on standardno posjeduje

Prilikom definiranja znakova mijenjamo samo adresu znakova upotrebom naredbe POKE »Oraz« posjeduje i jednu BASIC-naredbu koja uključuje potrebu za naredbom POKE. To je naredba CHAR iza koje slijede dva parametra. Prvi parametar određuje koji set znakove mijenjamo i može imati vrijednost od 0 do 2. Nula označava brojeve i znakove jednake velika slova, a dvojka mala slova. Drugi parametar je adresa stranice memorije na koju smo smjestili novu definiciju znakova. Ako je vrijednost drugog parametra nula, postavlja se definicija znakova kao i za uključenja računala. Kao što vidite upotrebom ove naredbe ne možemo promijeniti definiciju korisničkih znakova. Prema tome, ako želimo ove znakove mijenjati moramo upotrijebiti naredbu POKE

IZLAZ I ULAZ

Prilikom rada mikroračunala odredena informacija iz vanjskog svijeta stižu do mikroračunala. To obično nazivamo ulazom informacija. Ulaz informacija je s tastature ili s kasetofona. Vidjeli smo već da program može zahtijevat unos podataka preko tastature upotrebom naredbe INPUT. Kad na de na naredbu INPUT računalo stane i čeke da korisnik otipka neku vrijednost i pritisne tipku (CR). Tek potom se nastavlja izvođenje programa. U nekim programima potrebno je pročitati da li je na tastaturi pritisnuta neka tipka a ako nije normalno nastaviti izvođenja programa bez zaustavljanja. U tu svrhu rabimo izraz

INKEY A$

Naliskom na ovu naredbu računalo će pročitati tastaturu. Ako je pritisnuta neka tipka, string-varijabli A pridružit će se odgovarajući string, a u protivnom postavit će se duzina string-varijable A na nulu. Svakako se izvođenje programa normalno dalje nastavlja. U idućem programu automatska se ispisuje kod pritisnute tipke. Pripazite na liniju 20 u kojoj testiramo da li je uopće neka tipka pritisnuta.

```
10 INKEY A$
20 IF A$= " THEN GOTO 10
30 PRINT ASC(A$)
40 GOTO 10
```

Izlaz podataka iz računala najčešće je usmjeren k ekranu. Kako smo s ekranu dovoljno govorili pozabavit ćemo se malo ostalim »izlazima« mikroračunala.

Na sl. 1 vidimo da je konektor broj 4 predviđen za priključenje štampača. Štampač se priključuje prema protokolu RS232. To znači da na mikroračunalu »ORAO« možemo priključiti bilo koji štampač koji posjeduje interface RS232. Priključuju se samo tri voda onako kako je prikazano na sl. 4. Izlaz na štampač aktiviramo pritiskom na tipku (PF2) i upotrebom naredbe PTR. Ta naredba PTR dobiva parametar koji može biti 0 ili 1. Nula isključuje štampač, a jedinica ga uključuje. Nakon što uključimo štampač, sve što se ispisuje na ekran bit će automatski ispisano na štampač.

Protokol RS232 predvida više standardnih brzina komunikacije. Brzinu komunikacije u »orlu« određujemo naredbom RATE. Iza ove naredbe dolazi parametar koji ima vrijednost od 0 do 3 a znači ova brzina komuniciranja

0 – 300 baud
1 – 600 baud
2 – 1200 baud
3 – 2400 baud

Baud je jedinica koja označava brzinu prijenosa od jednog bita u sekundi. Prema tome 1200 bauda znači prijenos 1200 bita u sekundi. Treba imati na umu da će za svaki byte štampaču biti preneseno jedanaest bita. Od toga su osam bita podaci a tri bita za sinhronizaciju.

Izlaz iz računala je i zvučnik. BASIC mikroračunala »orao« ima naredbu za generiranje zvuka. To je naredba SOUND iza ove naredbe slijede dva parametra. Prvi parametar određuje visinu tona, a drugi parametar dužinu trajanja tona. Oba parametra mogu biti u rasponu od 0 do 255. Pri tome manji broj u prvom parametru predstavlja viši ton. Nula označava vrijednost 256 dakle, najdublji mogući ton. Što veću vrijednost ima drugi parametar duže će biti trajanje generiranog tona. I ovdje je nula vrijednost 256 dakle ton najdužeg trajanja.

Postoji samo jedan uređaj koji može biti i izlaz i ulaz za podatke. To je naravno kasetofon. Do sada sma već spomenuli dvije naredbe za rad s kasetofonom. To su naredbe SAVE i LOAD. Osim ovih postoje još

nekoliko naredbi za rad s kasetofonom. Kao prve, te su naredbe DMEM i LMEM. Ove naredbe služe za spremanje, odnosno učitavanje bloka memorije. Na primjer, ovaj

```
DMEM "SLIKA",24576,6192
```

spremit će na vrpcu sliku koju imamo na ekranu. Tekst u navodnicima označava ime pod kojim je slika spremljena. Prvi broj za imenom označava početnu adresu bloka u memoriji, a drugi broj ukupnu duljinu u bytesovima bloka koji spremamo na vrpcu. Navođenje oba broja je obavezno.

Izrazom

```
LMEM "SLIKA",24576
```

učitat ćemo spremljenu sliku s vrpce ponovno na ekran. Broj iza imena označava adresu na koju će se memorijski blok učitati. Ako želimo mozarne blok učitati na adresu različitu od one na koju je blok spremljen. Ako broj iza imena ne navedemo blok će se učitati na onu adresu s koje je bio spremljen. Ako na navedemo ime bloka učitat će se prvi blok koji nađe na vrpci.

Osim spremanja i čitanja programa i blokova memorija na vrpcu možemo smjestiti i podatke. Prije nego počnemo upisivanje podataka, moramo na vrpci «otvoriti» zapis u koji ćemo pisati. To se radi naredbom OPENW iza koje u navodnicima stavimo ime pod kojim spremamo podatke na vrpcu. Podatke na vrpcu nikon što smo otvorili zapis, upisujemo naredbom WRITE U svoj naredbi možemo navesti i numeričke i niz-varijable ... unutar jedne naredbe moguće je navesti više varijabli. Varijabla međusobnu odvajamo zarezima. Nakon što smo zapisali sve podatke, potrebno je zapis propisno zatvoriti. To radimo naredbom CLOSEW koja nema parametra. Evo programa koji zapisuje vrpcu brojeva od jedan do sto

```
10 OPEN "BLOK"
20 FOR I=1 TO 100
30 WRITE I
40 NEXT I
50 CLOSEW
```

Ovako spremljene podatke možemo učitati s vrpce...ali učitavati moramo u odgovarajuće polje podataka. To je zbog toga što «oko» ne može zaustaviti kasetofon pa bi mu podaci učitavani na neki drugi način «pobjegli» na vrpci.

Zapis koji želimo učitati moramo, naravno, prvo otvoriti. Za otvaranje zap sa služi naredbe OPENG iza ove naredbe u navodnicima dolazi ime bloka koji želimo otvoriti. Ako ne navedemo mêta unutar navodnika otvorit će se prvi blok koji nađe na vrpci. Podatke

čitamo naredbom INPUT. Pri tome moramo paziti da varijabla koju čitamo odgovara zapisu na vrpci: dakle ne smijemo čitati numeričku varijablu ako smo zapisali string i obratno. Poslije čitanja moramo zapis pravilno zatvoriti. To radimo naredbom CLOSEG koja nema parametar. Naredni program učitat će i ispisati na ekran zapis koji smo spremili s prethodnim programom. Učitava se u matricu pod imenom »A« koja ima sto elemenata.

```
10 DIM A(99)
20 OPENG "BLOK'
30 FOR I=0 TO 99
40 INPUT A(I)
50 NEXT I
60 CLOSEG
70 FOR I=0 TO 99
80 PRINT A(I);
90 NEXT I
```

Takvo zapisivanje na vrpcu mnogo je češće pri radu s diskovima. Ipak, dobro je da su se autori BASICA odlučili da omoguće ovakav rad na vrpci jer je takav način zapisivanja podataka vrlo često potreban u rijetki su korisnici mikroračunala »orao« kojima je dostupan disk.

POZIVI STROJNIH POTPROGRAMA

Unutar mikroračunala »orao« nalazi se veliki strojni program. Mikroračunalo »orao« ne razumije BASIC i taj strojni program je zaduženo da prevodi naredbe BASIC u informacije razumljive mikroprocesoru ugrađenom u »orao«. Pojedine dijelove tog programa možemo po volji pozivati iz BASICA odgovarajućim instrukcijama.

Za pozivanje strojnih programa predviđena je jedna funkcija i jedna naredba. Funkcija je USR i ima ovaj oblik

```
U=USR(U)
```

Pritom nije bitna vrijednost argumenta navedenog u izrazu a i sama funkcija ne daje nikakav rezultat. Umjesto toga izvršavanje ove instrukcije zazvat će

pozivanje strojnog programa čija je adresa definirana sadržajem memorijskih lokacije 4 i 5 Ne adresi 4 nalazi se niže a ne adresi 5 više byte 16-bitne adrese strojnog programa Tekvo pozivanje strojne rutine nije naročito praktično i suton BASICA zadržali su ga radi kompatibilnosti sa starijim verzijama BASICA

Mnogo je jednostavnije pozivanje upotrebom naredbe LNK Iza ove naredbe dolazi broj koji označave adresu rutine koju poz vamo

Prilikom pozivanje strojnih rutina treba biti oprezan i tocno znati što se poziva Neiskusnim korisnicima ne preporučujem eksperimentiranje s ov m instrukcijama Ne može nestati nikakve šteta na racunalu ali može se desiti da stradaju avi podaci koji se nalaze u memoriji, odnosno da se računalo mora resetirati r ponovno uključ ti u BASIC

Na samom početku knjige rekli smo kako startamo BASIC Isto tako rekli emo da se po ukljucenju računalo nalazi u monitoru Ako zelimo de se iz BASICA vratimo u monitor možemo to učinti dvojako prvo možemo pritisnuti tipku RESET na poleđini računala, a možemo upotr jebiti i naredbu EXIT Ova naredba nema parametra i nosi prijelaz iz BASICA u monitor O monitoru ćemo detaljno govoriti u narednom poglavlju

MONITOR

Mikroračunalo »oreo« opremljeno je izvrsnim dodatkom koji nam omogućuje rad u strojnom jezku Čim uključimo računalo nalazimo se u monitoru PROMPT monitora je zvjezd ca i ked nju ugledamo možemo se odlučit za ulazak u BASIC prema već opisanom postupku Isto tako možemo pozvati i BASIC bez uništavanje BASIC-programa odnosno bez brisanja memorije Ulazak u BASIC s brisanjem memorije naziva se hladni start BASICA, a ulazak bez brisanja memorije vrući start Vrući start izvodimo naredbom *BW

Da bismo mogli vruće startati BASIC potrebno je da je on već bio prethodno hladno startan te de nisu oštećene vnjednosti na nultoj stranici

Tipke reset vraća nas u monitor bez obzira na to kakav posao radi »oreo« Zbog toga možemo ovom tipkom prekinuti čak i beskonačne petlje u strojnom jeziku

Namjena je ove knjige da poslužI kao priručnik za učenje BASICA. Zbog toga ovdje ne može biti mnogo govora o strojnom programiranju. Tekst koji bi poslužio kao minimalan uvod u strojno programiranje bio bi vjerojatno po obujmu znatno veći od ukupne dužine teksta u ovoj knjizi. Oni koji žele naučiti strojno programiranje moraju se poslužiti nekom drugom literaturom. Kako je »one« građen oko mikroprocesora 6502 koji je vrlo raširan među mikroračunalima, napisano je mnogo naslova koji se bave problematikom programiranja mikroprocesora 6502. Zbog toga pretpostavljam da svi zainteresirani neće imati problema pri nabavljanju literature. Oni koji su se sada prv put sreli s računalom za sada neka rad u strojnom programiranju ostave za bolja vremena. Oni slobodno mogu čitanje nastaviti od početka sljedećeg poglavlja.

Za rad u strojnom jeziku stoji nam na raspolaganju sedam monitorskih komandi i minasembler. Sve adrese koje se unose u monitor, kao i sve vrijednosti koje unosimo moraju biti u heksadecimalnoj notaciji. Pogledajmo sada redom komande monitora.

Kao prvo to je disassembler koji pozivamo komandom

*Xnnnn

gdje nnnn označava heksadecimalnu adresu. Dissemitter je program koji prevara kod strojnog programa u mnemonike. Komandom X dobit ćete 20 redova disassembliranog teksta. Ako želimo sljedeću stranicu dovoljno je da unesemo samo

X

bez adrese i disassembliranje će se nastaviti tamo gdje je prekinuto.

Za brzu pretragu memorije možemo uzeti naredbu

*Ennnn mmmm

gdje su nnnn početne adrese, a mmmm završne adrese. Komanda E ispisivat će nam heksadecimalne sadržaje bajtova počevši od adrese nnnn završno s adresom mmmm. Ispisivat će se onaj vrijednosti u jednom radu. Ako želimo zaustaviti (ne prekinuti) tipkanje možemo pritisnuti bilo koju tipku. Ponovnim pritiskom ispisivanje će se nastaviti. U naredbi E navođenje adrese je obavezno.

Čitanje i mijenjanje memorijske lokacije obavit ćemo pomoću komande

*Mnnnn

gdje je nnnn heksadecimalna adresa. Komanda M će pisati će nam adresu i heksadecimalni sadržaj te lokacije. Sada možemo mnijeti heksadecimalnu vrijednost koja će se smjestiti u tu adresu. U obzir će se uzeti prve dva znaka koja navedemo pod pretpostavkom da pripadaju setu heksadecimalnih znakova. Nakon što pišemo novu vrijednost moramo pritisnuti tipku (CR). Ako ne želimo mijenjati vrijednost te lokacije dovoljno je da samo pritisnemo (CR) i ispisat će nam se adresa i sadržaj slijedeće lokacije. Ako se iz nekog razloga želimo vratiti na prethodnu lokaciju, pišemo umjesto nove vrijednosti znak minus (-) i pritisnuli tipku (CR). Kada želimo izaći iz M moda un jet ćemo bilo koj znak koji ne pripada setu heksadecimalnih znakova.

Ako ne navedemo adresu ze komande M komanda M će nzeti posljednju adresu s kojom je bila upotrijebljene ili neku adresu koja je preostala od ispitivanja drugih monitorskih komandi. Zbog mogućnosti da takvo oštetimo neki sadržaj koji nismo željeli mijenjati najbolje je da uvijek iza komande M navedemo adresu.

Komandom

***Cnnnn mmmm**

gdje nnnn označava početnu adresu a mmmm završnu adresu možemo zbrajati vrijednosti u bajtovima nekog memorijskog bloka. Ova nam komanda služi za provjeru sume nekog bloka i neodgovarajuća suma (u usporedbi s prijašnjim snimanjem) indicira nam da se sadržaj određenog bloka promijenio. Pri radu s kasetofonom to se može desiti zbog krivog učitavanje programa. Prema tome komandu C možemo upotrijebiti da ispitamo da li se program pravilno učitao.

Uz komandu C kao i uz komandu E navođenje adresa je obvezno.

Određeni memorijski blok možemo popuniti komandom

***Fnnnn mmmm ff**

gdje je nnnn označava početnu adresu nekog bloka, mmmm završnu a ff vrijednost koja se stavlja u sve bajtove tog bloka. Na primjer

***F6000 7FFF 00**

Izbrisat će cijel ekran jer će čitav video-RAM biti popunjen nulama. Komanda F svojn primjenu nalazi n radu s tablicama čije vrijednosti često treba »brisati« odnosno vraćati na nulu.

U komandi F potrebno je navest sva tri parametre. U protivnom, ova komende se neće izvršiti.

Vidjeli smo da pomoću komande E možemo ispisati blok memorije. Pritom su vrijednosti prikazane heksadecimalno. Vrlo sličnu funkciju ima i naredba

***hnnnn mmmm**

pri čemu nnnn označava početak bloka a mmmm kraj bloka koji želimo ispisati. Za razliku od komande E, ovdje će nam se ispisivati pet byte ova u jednom redu a tim da će byte-ovi biti najprije ispisani heksadecimalno a zatim će biti naprsani i odgovarajući znakovi ASCII. U navedenom primjeru ispisat će se sadržaj byte ova od 0f0f0 do 0f1F heksadecimalno

***H0F0 0f1F**

Svi znakovi izvan opsega 32 – 127 bit će zamijenjeni točkom. Ovakav način ispisivanja bloka memorije nužno nam je potrebni za brzo pregledavanje i korigiranje tablica u kojima se nalaze tekstualni podaci.

Osim izlistavanja i pretraživanja blokova memorije potrebno je moći pozvati strojnu rutinu. Ako želimo izvesti bespovratni skok na strojni program upotrijebit ćemo izraz

***Jnnnn**

gdje nnnn označava adresu na kojoj se nalazi strojna rutina koju pozivamo. Bespovratan skok znači da u trenutku poziva neće na stack biti smještena povratna adresa, za razliku od komande U o kojoj će kasnije biti govora.

Navikli smo da u svakodnevnom životu upotrebljavamo dekadski brojni sustav. Zbog toga nam je vrlo često teško baratat heksadecimalnim vrijednostima. U monitoru postoji komanda čija je funkcija pretvorba heksadecimalnih vrijednosti u decimalne. To je komanda

***#nnnn**

gdje nnnn označava heksadecimalnu vrijednost. Nakon ove komande, «enter» će nam ispisati decimalnu vrijednost broja navedenog u komandi

Dio memorije odnosno memorijski blok možemo preseliti na neko drugo mjesto naredbom

***Gnnnn mmmm iiii**

gdje je nnnn adresa na koju će se premjesti blok, mmmm je startna adresa bloka koji prenosimo a iii je završne adresa bloka koji prenosimo. Komanda G prilikom prijenosa se ne prenošenja podataka na neko drugo mjesto. Prenošenje programe ovom komandom moguće je jedino ako program ne sadržave niti jednu apsolutnu adresu. Svaka apsolutna adresa unutar programa ili bilo koji program koji je apsolutnom adresom pozivao program koji smo preselili neće pravilno raditi ako upotrijebimo ovu komandu

Komanda Q zahtijeva kao i komande F da se ne vadu sve tri potrebne parametre U protivnom komanda se neće izvršiti

Posljednje monitorska komanda je komanda

***Unnnn**

Ovdje nnnn označava adresu programa. Te komande služi za pozivanje strojnih programa, bez obzira na to da li su u pitanju programu u RAM u ili u ROM u

Komanda U zahtijeva navođenje adresa programa. Ovo su bile sve komande koje posjeduje monitor Osim monitora u mikroračunalo «orao» ugrađen je i miniassembler Pozivamo ga komandom

***Annnn**

gdje je nnnn adresa na koju će se generirati kod Naredbe se u miniassembler unose u standardnom formatu Sve brojne vrijednosti mereju biti u heksadecimalnoj notaciji Miniassembler ne prihvaća labele i svi skokovi moraju se navoditi numerički U relativnim skokovima navodimo apsolutnu adresu bajte u koji želimo skok U toku unošenja naredbe se odmah asembliraju teko da nam assembler ove trenutke ispisuje apsolutne adrese na koje će generirani kod Zbog toga nema klasične forme teksta, niti assembliranja završenog programa. Unatoč svim navedenim nedostacima ovaj miniassembler pruža velike mogućnosti i osobito je korisan kada želimo unijeti neku kraću strojnu rutinu Osnovna prednost je u tome što nas oslobađa bespotrebnog opterećivanja memorije assemblerskim programom

Ako pri pozivanju miniassemblera ne navedemo adresu miniassembler će uzeti adresu prestalu od neka prethodne monitorske instrukcije

Da izbjegnemo potrebu pisanje vrlo dugih programa možamo se služiti pozivanjem potprograma u ROM u

ŠTO DALJE?

Osnovne namjere pri radu na ovoj knjizi bile mi je da stvorim priručnik koji će pomoći početniku u prvom susretu s m kroračunalom Zbog toga veći dio knjige govori o osnovnim BASIC naredbama. Činjenice je da

nake BASIC naredba možda zaslužuju da se o njima više govori. Međutim ponavljam još jednom da se programiranje ne može naučiti ni iz etotinu knjiga. Jedini pravi put da se svlada tehnika programiranje jest da se radi sa računalom. Želio bih da ova knjiga pri tome bude vodič i pomoćnik.

Na kraju je potrebno istaknuti da je programski jezik BASIC namijenjen u prvom redu stjecanju prvih znanja o računalima i da bi svatko tko želi postati računarski pismene osobe morao svladati ovaj a potom preme svojim željama i potrebama i neki drugi programski jezik. Ima mnogo različitih teorija o tome koji bi programski jezik trebalo da bude »standard«. Mislim da bi programski jezik PASCAL mogao zadovoljiti većinu zainteresiranih da svladaju još neki viši programski jezik. Za ozbiljniji rad na programima nije dovoljno samo poznavanje programskog jezika potrebno je znati i mnogo detalja o samom stroju na kojem se radi. Oni koji žele izvući maksimum iz jednog mikroračunala moraju će svakako savladati i strojni jezik.

Nadam se da će vam znanje koje ste stekli iz knjige biti dovoljan poticaj da se samostalno uputite u istraživanje svijeta mikroračunala.

PROGRAMI

PRERAČUNAVANJE BROJEVA

```
 10 REM PRETVARANJE BROJEVA
 20 GOSUB 850
 30 DIM E$(15)
 40 FOR Q=0 TO 15
 50 READ E$(Q)
 60 NEXT Q
 70 REM IZBOR BAZE
 80 PRINT CHR$(4);CHR$(10);CHR$(
10);CHR$(10)
 90 PRINT " D - dec H - hex
   B - bin"
 100 PRINT CHR$(10);CHR$(10);CHR
$(10);CHR$(10)
 110 INKEY K$:IF K$="" THEN 110
 120 A=ASC(K$)-32
 130 IF A=66 GOTO 580
 140 IF A=72 GOTO 420
 150 IF A<>68 GOTO 110
 160 PRINT CHR$(10);"Unesi dec.
broj ';CHR$(10)
 170 GOSUB 730
 180 IF LEN(A$)=0 OR LEN(A$)>5 G
OTO 170
```

```
 190 FOR Q=1 TO LEN(A$)
 200 IF MID$(A$,Q,1)<"0" OR MID$
(A$,Q,1)>"9" GOTO 170
 210 NEXT Q
 220 D=VAL(A$)
 230 IF D>65535 GOTO 170
 240 H=D
 250 H$=""
 260 B$=""
 270 FOR Q=3 TO 0 STEP-1
 280 P=INT(H/16^Q)
 290 H=H-16^Q*P
 300 B$=B$+E$(P)
 310 P=P+48
 320 IF P>57 THEN P=P+7
 330 H$=H$+CHR$(P)
 340 NEXT Q
 350 REM ISPIS REZULTATA
 360 PRINT CHR$(13);CHR$(10);CHR
$(10);CHR$(10);CHR$(10);"DEC -";
D
 370 PRINT CHR$(10);"HEX - ";H$
 380 PRINT CHR$(10);"BIN - ";LEF
T$(B$,Q);"";RIGHT$(B$,8)
 390 PRINT CHR$(10)
 400 GOTO 70
 410 REM POTPROGRAM ZA HEX
 420 PRINT CHR$(10);"Unesi hex.
broj";CHR$(10)
 430 GOSUB 730
 440 IF LEN(A$)=0 OR LEN(A$)>4 G
OTO 430
 450 FOR Q=1 TO LEN(A$)
 460 IF MID$(A$,Q,1)>="0" AND MI
D$(A$,Q,1)<="9" GOTO 480
 470 IF MID$(A$,Q,1)<"A" OR MID$
(A$,Q,1)>"F" GOTO 430
 480 NEXT Q
 490 D=0:FOR Q=1 TO LEN(A$)
 500 P=ASC(MID$(A$,Q,1))
 510 P=P-48
 520 IF P>9 THEN P=P-7
 530 P1=LEN(A$)-Q
 540 D=D+P*16^P1
 550 NEXT Q
 560 GOTO 240
 570 REM POTPROGRAM ZA BIN
 580 PRINT CHR$(10);"Unesi bin.
broj";CHR$(10)
 590 GOSUB 730
 600 IF LEN(A$)=0 OR LEN(A$)>16
GOTO 590
 610 FOR Q=1 TO LEN(A$)
 620 IF MID$(A$,Q,1)>"1" OR MID$
```

```
(A$,Q,1)<"0" GOTO 590
 630 NEXT Q
 640 D=0
 650 FOR Q=1 TO LEN(A$)
 660 P=VAL(MID$(A$,Q,1))
 670 P1=LEN(A$)-Q
 680 D=D+P*2^P1
 690 NEXT Q
 700 D=INT(D)
 710 GOTO 240
 720 REM ISPITIVANJE TASTATURE
 730 PRINT CHR$(6);
 740 A$=""
 750 PRINT CHR$(13);A$;CHR$(5);
 760 INKEY K$:IF K$<>"" THEN 760
 770 INKEY K$:IF K$="" THEN 770
 780 A=ABC(K$)
 790 IF A=13 THEN RETURN
 800 IF A=31 AND LEN (A$)=1 THEN
A$=""
 810 IF A=31 AND LEN(A$)>1 THEN
A$=LEFT$(A$,LEN(A$)-1)
 820 IF A>96 AND A<103 THEN A=A-
32
 830 IF A<48 OR A>70 GOTO 750
 840 A$=A$+CHR$(A):GOTO 750
 850 PAINT CHR$(12)
 860 PRINT CHR$(10);CHR$(10)
 870 PRINT "        PRETVARANJE B
ROJEVA';CHR$(10);CHR$(10)
 880 PRINT "              PRIJE NEGO
UNESETE BROJ"
 890 PRINT "PAITISNITE D ZA DECI
MALNI, H  ZA"
 900 PRINT "HEKSADECIMALNI ILI B
ZA  BINARNI"
 910 PRINT "BROJ. UNESITE BROJ I
  PRITISNITE"
 920 PRINT "(CR).  ZA  NOVI  BRO
J  IZABERITE"
 930 PRINT "PONOVO D, H ILI B."
 940 PRINT CHR$(10);CHR$(10);CHR
$(10)
 950 PRINT "PRITSNITE (CA)"
 960 GOSUB 730
 970 PRINT CHR$(12)
 980 RETURN
 990 REM PODACI
 1000 DATA 0000,0001,0010,0011,0
100,0101,0110,0111
 1010 DATA 1000,1001,1010,1011,1
100,1101,1110,1111
```

GENERIRANJE ZVUKA

```
 10 REM GENERIRANJE ZVUKA
 20 REM UPIS STROJNOG PROGRAMA
 30 FOR Q=6000 TO 6026
 40 READ A:POKE Q,A
 50 NEXT Q
 60 REM POZIV STROJNOG PROGRAMA
 70 CLS
 80 PRINT"PRITISNI BILO KOJU TIP
KU ZA KRAJ"
 90 LNK 6000
 100 REM PODACI
 110 DATA 160,0,136,200,0,141,0,
136,24,144,245,162,255,202,208
 120 DATA 242,141,0,136,32,176,2
29,144,243,24,96,234
```

PROGRAM ZA CRTANJE

```
 10 REM PROGRAM ZA CRTANJE
 20 POKE 534,68
 30 POKE 535,3
 40 GOSUB 600
 50 POKE 228,127
 60 POKE 229,127
 70 X1=127
 80 Y1=127
 90 M=2
 100 MODE 255
 110 X=PEEK(228)
 120 Y=PEEK(229)
 130 PLOT X,Y
 140 POKE 248,0:LNK 768
 150 A=PEEK(248)
 160 PLOT X,Y
 170 IF M=2 AND A=0 GOTO 100
 180 MODE M
 190 IF A=32 THEN X1=X:Y1=Y:GOTO
 100
 200 IF(A<48 OR A>57) AND M<>2 T
HEN PLOT X,Y:GOTO 100
 210 IF(A<48 OR A>57) AND M=2 GO
TO 100
 220 IF A=51 THEN M=255
 230 IF A=48 THEN M=2
 240 IF A=49 THEN M=0
 250 IF A=50 THEN M=1
 260 MODE 0
 270 IF A<52 THEN PRINT CHR$(7);
:GOTO 100
 280 IF A=52 THEN MOVE X1,Y1:DRA
W X,Y:GOTO 100
 290 IF A=53 GOTO 340
 300 IF A=54 GOTO 370
 310 IF A=56 THEN CLS:GOTO 50
 320 IF A=55 GOTO 520
 330 POKE 535,231:POKE 534,28:EN
D
 340 MOVE X1,Y1:DRAW X1,Y
 350 DRAW X,Y
 360 DRAW X,Y1
 370 DRAW X1,Y1
 380 GOTO 490
 390 L=ABS(X1-X)
 400 H=ABS(Y1-Y)
 410 R=SQR(L*L+H*H)
 420 CIR X1,Y1,R
 490 POKE 228,X
 500 POKE 229,Y
 510 GOTO 100
```

```
520 POKE 233,INT(X/8)
530 POKE 232,31-INT(Y/8)
540 PRINT CHR$(11);CHR$(10);
560 LNK 59164
570 A=PEEK(252)
580 IF A=13 THEN GOTO 100
590 GOTO 560                    23
600 CLS
610 FOR Q=768 TO 843
 620 READ A
 630 POKE Q,A
 640 NEXT Q
 650 DATA 32,176,229,144,6,133,2
40,201,97,208,3,198,228,96,201
 660 DATA 100,208,3,230,228,96,2
01,119,208,2,230,229,201
 670 DATA 120,208,3,198,229,96,2
01,113,208,5,198,228,230
 680 DATA 229,96,201,101,208,5,2
30,228,230,229,96,201,121
 690 DATA 208,5,198,228,198,229,
96,201,99,208,251,230,228
 700 DATA 198,229,96,32,176,229,
133,252,96
 710 PRINT "    U programu kor
istimo  dva"
720 PRINT "kursora. Vidljivi po
```

```
kreæemo  po"
 730 PRINT "ekranu slovima Q,W,E
,A,D,Y,X,C."
 740 PRINT "Nevidljivi kursor pr
itiskoa  na"
 750 PRINT "razmaknicu smještamo
 na vidlji-"
 760 PRINT "vi kursor. Linija,
pravokutnik"
 770 PRINT "i krug koriste oba k
ursora.     Ostala komanda su:'
 780 PRINT:PRINT "    0 - KRETANJ
E"
 790 PRINT:PRINT "    1 - CRTANJE
"
 800 PRINT:PRINT "    2 - BRISANJ
E"
 810 PRINT:PRINT "    3 - INVERTI
RANJE"
 820 PRINT:PRINT "    4 - LINIJA"
 830 PRINT:PRINT "    5 - PRAVOKU
TNIK"
 840 PRINT:PRINT "    6 - KRUG"
 850 PRINT:PRINT "    7 - PISANJE
 (<CR> za izlaz)"
 860 PRINT:PRINT "    8 - BRISANJ
E EKRANA"
```

```
 870 PRINT:PRINT "   9 - IZLAZ I
Z PROGRAMA"
 880 PRINT:PRINT:PRINT "pritisni
 bilo koju tipku";
 890 INKEY A$:IF A$="" THEN 890
 900 CLS:RETURN
```

POREZ (IGRA)

```
  10 GOSUB 900
  20 PRINT "    ZDRAVO!    Ja uzima
m za porez"
  25 PRINT "brojeve s kojima je d
jeljiv brojkoji ti uzmeš."
  30 CLEAR : GOSUB 810
  40 GOSUB 120
  50 GOSUB 330
  60 GOSUB 420
  70 IF N1=0 GOTO 100
  80 GOSUB 530
  90 IF N1>1 GOTO 40
 100 GOSUB 690
 110 GOTO 750
 120 PRINT : PRINT
 130 PRINT "Ti uzimaš? ";
 140 INPUT K
 150 K=INT(K)
 160 IF K>0 AND K<=N GOTO 190
 170 PRINT : PRINT "Taj broj nij
e na listi!"
 180 GOTO 130
 190 IF L(K)=0 GOTO 170
 200 IF K>1 GOTO 230
 210 PRINT : PRINT "broj";K;"ni
ne daje za porez!"
```

```
 220 GOTO 130
 230 M=0
 240 FOR I=1 TO K/2
 250 IF K<>I*INT(K/I) OR L(I)=0
GOTO 290
 260 M=M+1
 270 T(M)=I
 280 L(I)=0
 290 NEXT I
 300 IF M=0 GOTO 210
 310 L(K)=0
 320 RETURN
 330 Y=Y+K
 340 PRINT : PRINT "za porez uzi
mam";
 350 FOR I=1 TO M
 360 PRINT T(I);
 370 Z=Z+T(I)
 380 NEXT I
 390 N1=N1-M-1
 400 PRINT : PRINT 'Moj iznos:";
Z;"Tvoj iznos:";Y
 410 RETURN
 420 REM ISPIS LISTE
 430 PRINT : PRINT "NOVA LISTA';
 440 IF N1=0 GOTO 510
 470 FOR I=1 TO N
 480 IF L(I)=0 GOTO 500
 490 PRINT I;
 500 NEXT I
 510 PRINT
 520 RETURN
 530 FOR I=N2 TO 1 STEP -1
 540 IF L(I)=0 GOTO 620
 550 FOR J=2 TO I/2
 560 IF L(J)=0 OR I<>J*INT(I/J)
GOTO 600
 570 N2=I
 580 M=1
 590 RETURN
 600 NEXT J
 610 M1=M1+1
 620 NEXT I
 630 PRINT : PRINT : PRINT "
 Za poraz uzimam preostala,"
 640 PRINT "zato što nema viša b
rojeva koji"
 645 PRINT "su djelijivi nekim od
 preostalih brojeva."
 650 Z=Z+M1
 660 PRINT : PRINT "Moj iznos ja
:";Z
 670 N1=0
 680 RETURN
 690 PRINT : PRINT "Tvoj iznos";Y
 700 IF Z>Y GOTO 730
```

```
 710 PRINT : PRINT "TI SI POBJED
IO!"
 720 RETURN
 730 PRINT : PRINT "JA SAM POBJE
DIO!"
 740 RETURN
 750 PRINT : PRINT "Želiš li nov
u igru?"
 760 INPUT A$
 780 IF LEFT$(A$,1)="N" THEN END

 790 PRINT CHR$(12);
 800 GOTO 30
 810 PRINT : PRINT "Koliko broje
va želiš na listi?"
 820 INPUT N
 830 N=INT(N)
 840 N2=N
 850 N1=0
 860 IF N<=0 GOTO 820
 870 IF N<=50 GOTO 900

 880 PRINT "Najviše 50 brojeva."

 890 GOTO 810
 900 DIM L(50)
 910 DIM T(10)
 920 Y=0
 930 Z=0
 940 M1=0
 950 PRINT : PRINT "LISTA JE";
 960 FOR I=1 TO N
 970 PRINT I;
 980 L(I)=1
 990 NEXT I
 1000 IF N>1 GOTO 1050
 1010 PRINT : PRINT "Velikodušno
od tebe,"
 1020 PRINT "sve za poraz!"
 1030 Z=1:Y=0
 1040 GOSUB 690 : GOTO 750
 1050 RETURN
 1060 END
 9000 PRINT CHR$(12);
 9010 PRINT "                PO
RE Z"
 : PRINT
 9020 PRINT "   Nakon što izaber
emo veličinu"
 9030 PRINT "liste (broj brojeva
u igri od 1"
 9040 PRINT "do 50), potrebno je
s te liste"
 9050 PRINT "uzimati brojeve."
 9060 PRINT " Brojeve uzimamo je
dan po jedan"
 9070 PRINT "a svaki put kada  u
zmemo  jedan"
```

```
 9080 PRINT "broj, 'orso' uzies
sve  brojeve"
 9090 PRINT "s kojims je tsj  br
oj  djeljiv."
 9100 PRINT "Nije dozvoljeno uzs
ti broj koji"
 9110 PRINT "nije djeljiv ni s j
ednim brojem"
 9120 PRINT "od brojevs preostsl
ih ns listi."
 9130 PRINT "Ako ostanu samo tak
vi  brojevi,"
 9140 PRINT "'orso'  uzims  sve
 preostalm."
 9150 PRINT "Svi  se  brojevi  k
oje   uzmemo"
 9160 PRINT "zbrajaju. Potrebno
je ns  krsju"
 9170 PRINT "iesti više nego što
 je  'orao "
 9180 PRINT "uzeo za porez."
 9190 PRINT : PRINT
 9200 RETURN
```

DODACI

U prilogu knjige naći ćete četiri tablice. Prve je popis svih naredbi koje prepoznaje »orlov« BASIC. O toj tablici već je bilo govora u tekstu.

U tablici broj 2 nalazi se set znakova koji postoje u »orlu« i njihovi kodovi. Kodovi znakova pridruženi su prema standardu ASCII, a tim što su unijete izmjene zbog potrebe dodavanja naših slova.

Tablica broj 3 sadržave popis grešaka koje prijevijuje »orao« i njihovo tumačenje.

Na kraju je dodana i tablica naredbi za mikroprocesor 6502 i učinak tih naredbi na status registar. U toj tablici flegovi su označeni ovim kraticama

N – NEGATIVE
Z – ZERO
C – CARRY
I – IRQ DISSABLE
D – DECIMAL MOD
V – OVERFLOW

Za označavanje učinka pojedine instrukcije upotrijebljeni su ovi znakovi

* – fleg postavljen prema rezultatu instrukcije
– – fleg neizmijenjen nakon instrukcije
1 – fleg postavljen nakon instrukcije
0 – fleg očišćen nakon instrukcije
x – fleg sadržava sedmi bit testirane memorije
y – fleg sadržave šesti bit testirane memorije

Na kraju knjige dodano je i nekoliko BASIC-programa

TABLICA 1

^	EXP	POKE
*	FOR.. «NEXT	POS
+	FRE	PRINT
	GOSUB.. RETURN	READ
/	GOTO	REM
<	IF ... GOSUB	RESTORE
<=	IF ... GOTO	RIGHT$
<>	IF ... THEN	RND
=	INPUT	RUN
>	INT	SAVE

>=
ABS
AND
ASC
ATN
CHR$
CLEAR
CONT
COS
DATA
DEF FN
DIM
DRAW
END
LEFT$
LEN
LIST
LOAD
LOG
MID$
MOV
NEW
NOT
ON GOSUB
ON GOTO
OR
PEEK
PLOT
SIN
SIN
SPC
SQR
STOP
STR$
TAB
TAN
USR
VAL

TABLICA 2

32 -
33 - !
34 -
35 - #
36 - $
37 - %
38 - &
39 -
40 - (
41 -)
42 - *
43 - +
44 - ,
45 - -
46 - .
47 - /
48 - 0
49 - 1
50 - 2
51 - 3
52 - 4
53 - 5
54 - 6
55 - 7
56 - 8
57 - 9
58 - :
59 - ;
60 - <
61 - =
62 - >
63 - ?
64 - @
65 - A
66 - B
67 - C
68 - D
69 - E
70 - F
71 - G
72 - H
73 - I
74 - J
75 - K
76 - L
77 - M
78 - N
79 - O
80 - P
81 - Q
82 - R
83 - S
84 - T
85 - U
86 - V
87 - W
88 - X
89 - Y
90 - Z
91 - Č
92 - Ć
93 - Đ
94 - Š
95 - Ž
96 - ^
97 - a
98 - b
99 - c
100 - d
101 - e
102 - f
103 - g
104 - h
105 - i
106 - j
107 - k
108 - l
109 - m
110 - n
111 - o
112 - p
113 - q
114 - r
115 - s
116 - t
117 - u
118 - v
119 - w
120 - x
121 - y
122 - z
123 - č
124 - ć
125 - đ
126 - š
127 - ž

TABLICA 3

NF – NEXT bez prethodne naredbe FOR
SN – sintaktička greška
RG – RETURN bez prethodne naredbe GOSUB
OD – nedovoljno podataka ili više podataka READ nego DATA
FC – vrijednosti izven opsege (pri pozivu funkcije)
OV – rezultat izven opsega (u matematičkim operacijama)
OM – memorija je puna ili ima previše GOSUB nivoa
US – skok ili poziv nedefinirane naredbe
BS – pogrešan poziv pozvan nedefinirani element polje
DD – dvostruko dimenzionirana matrica
/0– dijeljenje s nulom
ID – nedozvoljena naredba upotrijebljena u komandnom modu
TM– pogrešan tip podataka (pri pozivu funkcije)
LS – prevelika dužina stringa
ST– zbroj previše složen
CN – CONT bez prethodnog STOP ili (CTL) C
UF – nije definirana funkcija

TABLICA 4

		N	Z	C	I	B	V
ADC	A=A+M+C	*	*	*	_	_	*
AND	A=A AND M	*	*	_	_	_	_
BEL		*	*	*	_	_	_
BCC	if C=0 then PC=PC+nn	_	_	_	_	_	_
BCS	if C=1 then PC=PC+nn	_	_	_	_	_	_
BEQ	if Z=1 then PC=PC+nn	_		_	_	_	_
BIT	A AND M, N=b7N V=b6M	N	*	_	_	_	V
BMI	if N=1 then PC=PC+nn	_		_	_	_	_
BNE	if Z=0 then PC=PC+nn	_		_	_	_	_
BPL	if N=0 then PC=PC+nn	_		_	_	_	_
BRK	PC=*brk	_		_	_	1	_
BVC	if V=0 then PC=PC+nn	_		_	_	_	_
BVS	if V=1 then PC=PC+nn	_		_	_	_	_
CLC	C=0	_		0	_	_	_
CLD	D=0	_		_	0	_	_
CLI	I=0	_		_	0	_	_
CLV	V=0	_		_	_	_	0
CMP	A-M	*	*	*	_	_	_

		N Z C I D V
CPX	X M	↕ ↕ ↕
CPY	Y-M	↕ ↕ ↕
DEC	M=M-1	↕ ↕
DEX	X=X-1	↕ ↕
DEY	Y=Y-1	↕ ↕
EOR	A=A EOR M	↕ ↕
INC	M=M+1	↕ ↕
INX	X=X+1	↕ ↕
INY	Y=Y+1	↕ ↕
JMP	PCL=(PC+1), PCH=(PC+2)	
JSR	PC to (SP), PCL=(PC+1), PCH=(PC+2)	
LDA	A=M	↕ ↕
LDX	X=M	↕ ↕
LDY	Y=M	↕ ↕
LSR		↕ ↕ ↕
NOP		
ORA	A=A OR M	↕ ↕
PHA	A to (SP)	
PHP	P to (SP)	
PLA	A from (SP)	↕ ↕
PLP	P from (SP)	↕ ↕ ↕ ↕ ↕ ↕
ROL		↕ ↕ ↕

		N Z C I D V
RTI	PC from (SP) P from (SP)	↕ ↕ ↕ ↕ ↕ ↕
RTS	PC from (SP)	
SBC	A=A-M-(C-1)	↕ ↕ ↕ ↕
SEC	C=1	
SED	D=1	
SEI	I=1	
STA	M=A	
STX	M=X	
STY	M=Y	
TAX	X=A	↕ ↕
TAY	Y=A	↕ ↕
TSX	X=S	↕ ↕
TXA	X=A	↕ ↕
TXS	X=S	↕ ↕
TYA	Y=A	↕ ↕

Popularno naučno tehničke biblioteke
KNJIGA DVADESET TREĆA

Damir Muraja, dipl. inž.
«ORAO – UVOD U RAD I PROGRAMIRANJE»

II izmijenjeno i dopunjeno izdanje

Glavni urednik
DUŠKO MALVIĆ

Tehnički urednik
VJEKOSLAV BOSNAR

Recenzenti
BOBKO BOKANIĆ, pref
BRANKO KEREČIN, dipl. inž.

Lektor
ZORKA HORVATIĆ

Pripremo za tisak
IZDAVAČKI ODJEL NARODNE TEHNIKE HRVATSKE
Zagreb Dalmatinska 12

Tisak
«Zadružna štampa» — OOUR tiskarska djelatnost Zagreb Ilica 35

GRADSKA KNJIŽNICA I ČITAONICA
METEL OŽEGOVIĆ VARAŽDIN

004.3
MUR
O
171785

GRADSKA
KNJIŽNICA I
ČITAONICA
METEL OŽEGOVIĆ
VARAŽDIN

SADRŽAJ

UVOD 3
ŠTO MOŽE «ORAO» 5
PRIKLJUČIVANJE 5
KAKO POČETI 7
TASTATURA 8
NEKI «STRUČNI» POJMOVI 10
ŠTO JE BASIC 11
PRIMJENA BASICA 12
PA DA POČNEMO! 13
KOMBINACIJE IZRAZA U PRINT NAREDBAMA 18
MOJ PRVI PROGRAM 18
ISPRAVLJANJE TEKSTA PROGRAMA (EDITOR) 23
PETLJA 25
UNOS PODATAKA (INPUT) 28
NAPOMENE U PROGRAMU (REM) 32
DONOŠENJE ODLUKA (IF) 34
NAREDBE ZA PREKIDANJE PROGRAMA 38
FUNKCIJA INT 39
SLUČAJNI BROJ (RND) 41
PISANJE PROGRAMA 43
Opis i pravila igre 43
Dijelovi programa 43
Skica programa 44
ČUVANJE PROGRAMA 46
POTPROGRAMI 51
PODACI UNUTAR PROGRAMA 53
NUMERIČKE FUNKCIJE 58
STRING-FUNKCIJE 61
PRINT-FUNKCIJE 65
KORISNIČKE FUNKCIJE 69
POLJE PODATAKA 70
LOGIČKE OPERACIJE 73
LOGIČKE OPERACIJE U BASICU 77
GRAFIKA 79
JOŠ JEDNOM O EKRANU 83
MEMORIJA 85
IZLAZ I ULAZ 88
POZIVI STROJNIH POTPROGRAMA 92
MONITOR 93
ŠTO DALJE? 97
PROGRAMI 98
Preračunavanje brojeva 98
Generiranje zvuka 101
Program za crtanje 102
Porez (igra) 104
DODACI 108
Popis naredbi u BASICU 108
Set i kodovi znakova (ASCII) 108
Popis grešaka 110
Popis naredbi za mikroprocesor 6502 110

GRADSKA
KNJIŽNICA I ČITAONICA
METEL OŽEROVIĆ
VARAŽDIN
004.3
MUR
O